Toni Sarvela

Jatkuvan integraation kehitys ja avoimen lähdekoodin lisenssienhallinta

Insinööri (AMK)

Tietotekniikka

Kevät 2018

KAJAANIN
AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

KAJAANIN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Tiivistelmä

Tekijä(t): Toni Sarvela

Työn nimi: Jatkuva integraation kehitys ja avoimen lähdekoodin lisenssienhallinta

Tutkintonimike: Insinööri (AMK), tietotekniikka

Asiasanat: jatkuva integraatio, testiautomaatio, lisenssit, avoin lähdekoodi

Tämä opinnäytetyö keskittyi kahteen pääaihealueeseen, jotka pyrittiin yhdistämään suuremmaksi kokonaisuudeksi; jatkuva integraatio sekä avoimen lähdekoodin lisenssienhallinta. Työn ajatus sai alkunsa työharjoitteluaikana, joka suoritettiin toimeksiantona Mecano Businessille.

Työn tavoitteena oli päivittää Mecanon CI-ympäristöä ja luoda tähän yhteyteen lisenssienhallinta-ratkaisu, jolla voidaan tunnistaa avoimen lähdekoodin lisenssejä versionhallinnasta ja tuoda informaatiota näistä kehittäjille. Eräät avoimen lähdekoodin lisenssit vaativat, että lähdekooditiedostoista käännetyt ohjelmistot sisältävät maininnan avoimen lähdekoodin kehittäjästä. Lisenssienhallinnan päätoimintona on siis luoda näistä lisensseistä tällainen tiedosto lisenssiehtojen mukaisesti. Lisenssienhallinta sopii hyvin jatkuvan integraation ratkaisun yhteyteen, koska tällä tavalla lisenssienhallinta on osana jokaisessa käännetyssä ohjelmistoversiossa ja tiedot lisensseistä pysyvät ajan tasalla.

CI-ympäristön päivittämisellä tarkoitetaan itse CI-työkalun päivittämistä uudempaan versionumeroon ja uuden palvelinverkoston käyttöönottamista uuden version kanssa. CI-työkalun päivittäminen mahdollisti uuden skriptikielen käyttämisen, jolla pystyttiin paremmin ohjaamaan ohjelmiston integraatiovaiheita, vaikka koko integraatioprosessi olisikin pitkä ja monimutkainen. Päivittämisen tuomat edut paransivat ohjelmistovalmiutta ja testauksen koordinointia uudessa palvelinverkostossa.

KAJAANIN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Abstract

Author(s): Toni Sarvela

Title of the Publication: Continuous Integration Development and Open Source Management

Degree Title: Bachelor of Engineering, Information Technology

Keywords: continuous integration, test automation, licenses, open source

This thesis focused on two primary areas, which were attempted to be united into one larger entirety; continuous integration and license management. The work got its start during the author's practical training, which was later carried out as a commissioned assignment to Mecano Business.

The objective of the thesis was to update the CI-environment and create a license management solution to this context, which could be used to identify open source licenses from repository and bring information of the findings to the developers. Some of the open source licenses demand that software includes some kind of a mention of the open source creator. The main feature of license management is to produce this mention according to the license terms. The license management fits well to the context of the continuous integration because this way the license management is part of every single compiled software version of the program and the license information stays up to date.

Updating the CI environment means updating the CI tool itself to a newer version number and introducing a new server network with the new version. Updating the CI tool enabled the use of a script language that allowed better control of software integration steps even if the whole integration process would be long and complicated. The benefits of updating improved software readiness and test coordination with the new server network.

Sisällys

Symboliluettelo

AJAX	Asynchronous JavaScript And XML
Back end	Ohjelmiston dataa käsittelevä kerros
CI	Continuous Integration
CSS	Cascading Style Sheets
Duplikaatti	Kopio toisesta olemassa olevasta kappaleesta
Front end	Ohjelmiston visuaalisia toimintoja ja käyttäjäsyötteitä ohjaava kerros
HTML	Hypertext Markup Language
Korruptointi	Tiedoston tai muistinpaikan kirjoittaminen sellaiseen muotoon, että ohjelma ei pysty enää lukemaan tätä normaalisti
PHP	Hypertest Preprocessor

1 Johdanto

Ketterä ohjelmistokehitys on yleistynyt ja tarjoaa vaihtoehdon perinteiselle vesiputousmallille tuotekehityksessä. Ketterillä menetelmillä tarkoitetaan tuotteen kehitysmenetelmää, jossa tuotteen suunnittelua ja -kehitystä tapahtuu projektin edetessä. Ketteryys (eli Agile) perustuukin tiimin kehittäjien, johdon, markkinoinnin sekä asiakkaiden väliseen kommunikointiin ja yhteistyöhön (1). Yksi agilen ideologiaa hyödyntävistä ohjelmistokehitystavoista on jatkuva integraatio. Jatkuvan integraation (continuous integration) toimintaperiaate perustuu kehittäjien integrointiajanjaksojen lyhentämiseen sekä ohjelmistojenkokonaisuuksien automatisoituun kokoamiseen ja testaamiseen. Jatkuvan integraation hyötynä on ohjelmistovalmiuden varmistaminen kehityksen aikana, jonka avulla taas voidaan havaita koodivirheet nopeammin ja reagoida niihin helpommin (2).

Moni yritys ympäri maailmaa on huomannut jatkuvan integraation hyödyt ja alkanut implementoida omaan ohjelmistokehitykseensä enemmän ketteriä menetelmiä. Tämän työn toimeksiantaja, Mecano Business, pyrkii myös kehittämään omaa jatkuvan integraation toteutustaan. Kehityksen kohteita ovat erityisesti automatisoidut testiympäristöt, jotka ovat osana tuotteen julkaisuputkea. Automaattisilla testeillä pystyttäisiin paremmin varmistumaan ohjelmistokokonaisuuksien toiminnasta ja nopeuttamaan tuotantoprosessia ja havaitsemaan virheet nopeammin. Lisäksi eräs tämän työn keskeisistä aiheista on lisenssienhallintajärjestelmän luominen ja liittäminen osaksi julkaisuputkea. Tämän toteutuksen tehtävänä on tunnistaa käännettävästä koodista mahdollisia vapaan lähdekoodin lisenssejä ja pitää kirjaa näistä. Ylöskirjatut tiedostot todennetaan myöhemmin järjestelmän käyttöliittymän kautta. Syynä tälle järjestelmälle on mahdollisten lisenssisopimusten rikkomisen välttäminen, esimerkiksi huolimattomuudesta johtuen.

Työn aiheen valinta liittyy Mecanon Businessin tarpeeseen kehittää tätä ympäristöä sekä aiheen monipuolisuuteen. Tämän aiheen toteutus vaatii hyvää ymmärrystä eri rajapintojen ja kehitysympäristöjen välisestä keskustelusta, mikä toimii tästä johtuen erinomaisena oppimisympäristönä. Pelimoottoriohjelmoinnin eräs keskeisimmistä asioista on suorituskyvyn varmistaminen, joka voidaan saavuttaa ainoastaan, jos ohjelmiston matalan tason komponentit ovat hyvin rakennettuna. Työn toteutukseen liittyvät vahvasti Linux-ympäristöt, jotka antavat hyvän ymmärryksen siitä, mitä tietokoneen matalalla tasolla oikeasti tapahtuu verrattuna perinteiseen Windows-ympäristöön ja C++-kehitykseen.

2 Työn teoreettinen tausta

2.1 Ketterä ohjelmistokehitys

Ohjelmistokehitystä on tavallisesti suoritettu kahdella eri tapaa; perinteisellä vesiputousmallilla ja ketterillä menetelmillä. Perinteinen vesiputous malli on saanut nimensä sekä ’perinteisen’-arvonimensä ollessaan ensimmäisiä ohjelmistokehitysmalleja. Koko ohjelmistokehitysprosessi tapahtui vaiheittain aina ohjelmiston suunnittelusta julkaisuun asti. Prosessin vaiheet kestivät pitkään, eikä vaiheista palauduttu tavallisesti kovin pitkäksi aikaa aiempiin vaiheisiin. Tämä malli ei kuitenkaan kovin usein toiminut useimmilla yrityksillä. Ohjelmiston suunnitelmat ja aikataulut on rajattu vesiputousmallissa erittäin tarkkaan, joten kehitystyössä ei ole kovin paljon varaa ongelmatilanteille. (3)

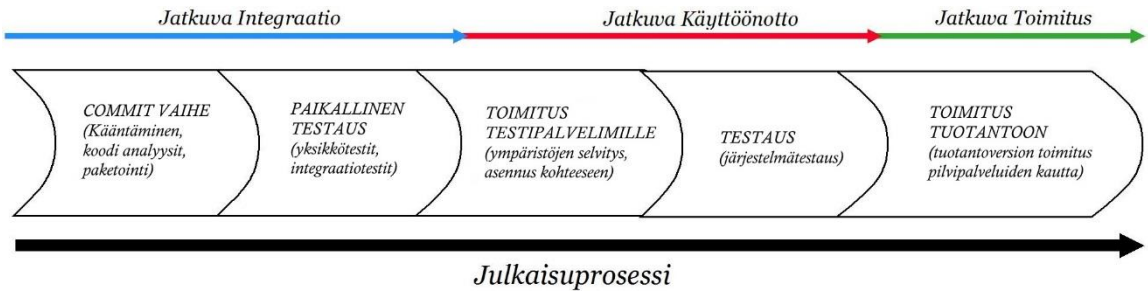
Ohjelmistokehitys ei tavallisesti ole kuitenkaan kovin suoraviivaista. Ohjelmistokehitystä on todella vaikea mitata työmäärältään, koska kirjoitettujen koodirivien määrä harvoin kuvastaa tehtyä työmäärää ohjelmistoon. Viikoittain ilmenee ongelmatilanteita, joista on hyvä käydä keskustelua muiden kehittäjien kanssa. Tätä varten kehitettiin ketteriä menetelmiä, joilla pyrittiin kuvaamaan ohjelmistokehitystä jatkuvana suunnittelu-, kehitys- ja testausprosessina. Ketterälle kehitykselle on ominaista pitää projektin suunnittelu ennen kehitysvaiheita mahdollisimman avoimena keskittämättä suunnittelua yksityiskohtiin liikaa. (4)

Scrum on yleisesti käytetty ketterän ohjelmistokehityksen malli, jossa kehitys toteutetaan noin 2 - 4 viikon sykleissä eli sprinteissä. Sprintti aloitetaan aina tekemällä yleiskatsaus kehityskohteesta ja suunnittelemalla tämän perusteella kehityksen suunnan ja kehitettävät aiheet sprintin kehitysvaiheelle. Aiheet pyritään pitämään mahdollisimman pieninä, mutta kehitettävät ominaisuudet suunnitellaan yksityiskohtaisesti. Kehitettävistä ominaisuuksista tehdään tavallisesti sprint backlog, joka on yksinkertainen tarkistuslista kehitettävistä ominaisuuksista. Nämä ominaisuudet pyritään kehittämään sprintin aikana, jonka jälkeen voidaan suorittaa taas sprintin jälkikatsaus ja aloitetaan seuraavan sprintin suunnittelu. Kuvassa 1 on esitetty scrumin sprintin kulku, jonka lopputuloksena aina mahdollisesti julkaistava tuote. (5,6)

Kuva 1. Scrumin kehitysjakson esitys. Scrum on tunnetuin ketterä ohjelmistokehitysmetelmä (3)

2.2 Julkaisuputki

Automatisoitua prosessiketjua, jonka alussa on versionhallinnan sisältö ja lopussa valmis julkaistava ohjelmistokokonaisuus, kutsutaan julkaisuputkeksi. Julkaisuputki koostuu jatkuvan integraation tuotoksesta, ohjelmiston testaamisesta ja jonkinlaisesta toteutuksesta ohjelmistokomponentin tai – kokonaisuuden julkaisemista asiakkaan käyttöön. Se, mitä vaiheita julkaisuputki sisältää, riippuu täysin esimerkiksi kehittäjistä, kehitysympäristöistä, yrityksen palvelinarkkitehtuurista sekä kehitettävästä tuotteesta. Jokaiselle yritykselle sopivan julkaisuputken rakentaminen on todennäköisesti mahdotonta. Kuvassa 2 on Julkaisuputken vaiheiden esitys. (7)



Kuva 2. Julkaisuprosessin kuvaus käyttäen hyväksi automatisoituja menetelmiä

Ohjelmistotuotekehityksessä julkaisuprosessin vaiheet saattavat vaihdella monista eri syistä, kuten kehittäjien testaamisen painoarvosta. Koko prosessin alku ja loppu ovat kuitenkin aina samat, eli kehitettävän tuotteen saattaminen kehittäjiltä asiakkaan käyttöön.

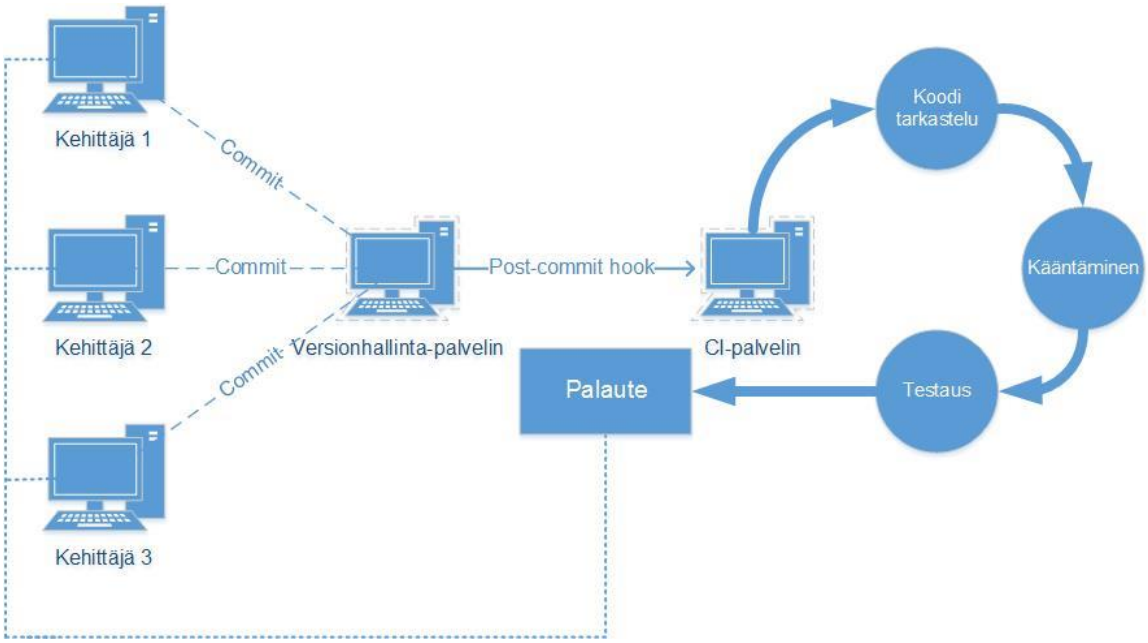
Julkaisuputki on julkaisuprosessin vaiheiden automatisointia, joka parhaimmillaan kattaa koko välin kehittäjän koodimuutoksesta asiakkaan käytössä olevaan ohjelmistoon. Tällaisen putken implementointi omaan yritystoimintaan on iso askel, mutta automatisointia voi lähteä toteuttamaan vaiheittain, aloittaen jatkuvan integraation menetelmistä ja etenemällä systemaattisesti kohti julkaisemista.

Julkaisuputken kehittäminen vähentää julkaisuun vaadittavien resurssien käyttöä ja parantaa palvelun laatua käyttämällä hyödykseen ohjelmistotestausta sekä palvelinverkostoja. Tämä vapauttaa kehittäjät toisteisten töiden parista ja antaa täten lisää aikaa kehitystehtäviin. Ohjelmiston integrointi saattaa olla erittäinkin pitkä lista virhealttiita manuaalisia tehtäviä, jotka voidaan helposti hoitaa automatisoidusti käyttäen hyödyksi erilaisia palvelimia. (8)

2.3 Jatkuva Integraatio

Perinteinen jatkuvan integraation toteutus vaatii vähintään kaksi palvelinta. Ensimmäinen palvelin suorittaa versionhallintaa ja toinen CI-ympäristöä. Useimmilla yrityksillä nämä palvelimet pyörivät eri koneilla, mutta palvelimet voivat olla myös samalla koneella. Suuremmissa projektikokonaisuuksissa voi kuitenkin aiheutua suoritusongelmia, jos palvelimet pyörivät samalla koneella ja ohjelmiston kääntämiseen menee paljon aikaa.

Jatkuva integraatio on siis yksinkertaisuudessaan laitteistojen ja ohjelmistojen välinen prosessiketju, joka alkaa kehittäjän luovuttaessa tekemänsä muutokset versionhallintaan omasta paikallisesta kopiostaan. Kehittäjien kirjoittamat koodit eivät siirry todellisuudessa versionhallintapalvelimelle kokonaisina tiedostoina, vaan pikemminkin muutoksina, jotka voidaan integroida päähaaraan (9). Tämän metodin hyödyt ovat suuret, koska sillä voidaan vähentää kehittäjien ja versionhallintapalvelimien välistä tietoliikennettä, pitää kirjaa muutosten välisistä ristiriidoista sekä tarjota tapa palauttaa päähaaran lähdekoodit aiempaan muotoonsa. Kuvassa 3 on yksityiskohtaisempi esitys julkaisuputken commit vaiheesta.



Kuva 3. Jatkuvan integraation perinteinen toimintaperiaate

Jatkuvan integraation toimintaperiaatteiden mukaan versionhallintapalvelin lähettää viestin käännösympäristöä suorittavalle palvelimelle, kun versionhallintaan luovutetaan koodimuutoksia. Tätä ympäristöä kutsutaan yleisesti CI-palvelimeksi. (9)

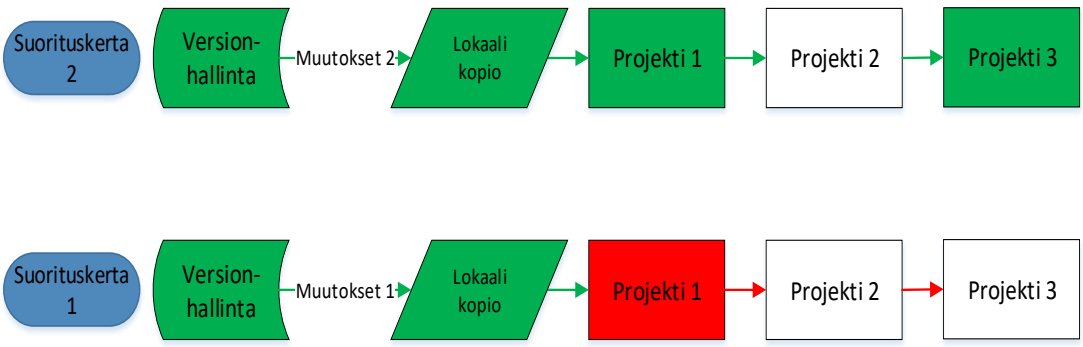
CI-palvelimen tehtävänä on rakentaa versionhallintapalvelimelta saaduista lähdekoodeista ohjelmistokokonaisuus, joka on suoritusvalmiina palvelinta ylläpitävällä koneella. Rakentamisella tarkoitetaan kääntämistä, yksikkötestausta, integraatiotestausta ja laadunvarmistusanalyysiä (7). Tämä vaatii, että koneella on lähdekoodien kääntämistä varten kääntäjä sekä testaamista varten skripti tai ohjelma, jolla testit voidaan suorittaa. Rakennuksen jälkeen CI-palvelimelta viedään viesti takaisin kehittäjille operaation onnistumisesta tai epäonnistumisesta.

Jatkuva integraatio mahdollistaa ohjelmiston julkaisuvalmiuden tarkastelun koko kehitysvaiheen ajan. Jatkuvan integraation suoritettavat operaatiot ovatkin tavallisesti samat kuin julkaistavaa ohjelmistokoostetta luotaessa. Äärimmillään jatkuvan integraation operaatiot tuottavat täysin testatun sekä asennusvalmiin ohjelmistopaketin, joka voidaan julkaista erilaisten digipalveluiden kautta tai jopa laittaa levylle valmiiksi asennusta varten. (8)

2.3.1 Nopeat ohjelmistokoosteet

Jatkuvan integraation on tarkoitus olla nopea, joten sen toteutuksessa on tarkoitus myös ohjata koosteen luominen siihen suuntaan, että jatkuvassa integraatiossa käytettäisiin mahdollisimman paljon valmiiksi luotua materiaalia. Jatkuvan integraation putkessa on turha aloittaa aina puhtaalta pöydältä jokaisella suorituskerralla. Tämä tarkoittaa käytännössä, että versionhallinnasta ei haluta hakea kaikkia lähdekoodeja uudestaan vaan pelkästään muutokset. Lähdekoodien uudelleen hakeminen tarkoittaisi myös sitä, että käännettyjen lähdekoodien binääritiedostot pyyhittäisiin samalla paikallisen kopion kansiorakenteista. Tällä tavalla kääntäjät osaavat käyttää valmiiksi luotuja binääritiedostoja koosteen tekemisessä, joka säästää huomattavan määrän aikaa jokaisella suorituskerralla. Tämä koskeen tietenkin vain käännettäviä ohjelmistokieliä.

Koosteen luomista voitaisiin nopeuttaa vielä enemmän, jos jatkuvan integraation toteutus olisi dynaamisempi, ja ei pyrittäisi kääntämään kaikkia komponentteja aina uudestaan ottamatta selvää ensin onko komponenttiin tehty edes muutoksia sitten viime kerran. Tämän toteuttaminen hyvin on kuitenkin erittäin hankalaa ja virhealtista. Putki ei aina pysähdy ohjelmointivirheisiin lähdekoodissa, vaan virheitä saattaa olla esimerkiksi itse putken toteutuksen komentosarjoissa tai putkessa käytettävissä tietokoneissa, joten systeemin pitäisi aina olla tietoinen siitä onko lähdekoodeja edes pyritty kääntämään aiemmilla suorituskerroilla. Kuvassa 4 on graafisesti havainnollistettu tällainen ongelma. Kuvan malliesimerkissä on kaksi suorituskertaa, joista molemmilla kerroilla muutosten siirtäminen onnistuu versionhallinnasta lokaaliin kopioon. Muutokset 1 sisältää muutoksia, jotka vaikuttavat projektiin 1 ja projektiin 2. Tällä suorituskerralla nämä projektit tulisi siis kääntää molemmat, mutta putki epäonnistuu projektia 1 kääntäessä, joten projekti 2 jää kääntämättä. Suorituskerralla 2 virheet on korjattu projektiin 1 ja tämän lisäksi muutokset vaikuttavat myös projektiin 3. Tällä suorituskerralla projektiin 2 ei kuitenkaan ole tullut muutoksia eikä putkeen ei ole luotu tapaa, jolla kääntämättömäksi jääneet projektit siirtyisivät seuraavalle suorituskerralle, joten muutokset projektiin 2 jää kääntämättä uusimpaan koosteeseen.



Kuva 4. Kääntämisongelma automatisoidussa putkessa

CI-palvelimen lokaali kopio olisi kuitenkin hyvä välillä siistiä varmistaakseen, että binääritiedostot koostuvat aina uusimmista lähdekoodeistakoodeista eikä koodeissa ole riippuvuuksia, joita putkessa ei pystytä havaitsemaan, mikäli lokaalissa kopiossa on jo valmiita binääritiedostoja. Paras tapa olisi, jos ”siistivä suorituskerta” ajoitettaisiin ns. konttoriajan ulkopuolelle, jolloin muutoksia versionhallintaan ei tule ja putki voi kestää kauemmin mitä konttoriaikana. Tämä suoritus kerta voisi olla esimerkiksi yöllä, jolloin binääritiedostot olisivat valmiina aamulla uuteen päivään ja uusiin muutoksiin versionhallintaan.

2.4 Ohjelmistotestaus

2.4.1 Automatisoitu testaaminen

Automatisoidulla testaamisella pyritään siis varmistamaan ohjelmiston toiminta tekemällä testejä. Automatisoidun testauksen tarkoituksena ei ole tuotteen kaiken kattava testausympäristö, jonka tuloksena voidaan määritellä ohjelmistoversion käyttökelpoisuus asiakkaan käytössä. Automatisoidulla testaamisella pyritään vapauttamaan ihmistestaajat datapainotteisista itseään toistavista testitapauksista, jotka eivät huomioon käyttäjien psykologista puolta ohjelmiston käytettävyydessä. Automaattisilla testeillä pyritään havaitsemaan tilanteita, joissa ohjelma esimerkiksi kaatuu tai että ohjelman toiminnot eivät toimi oikein joillain tietyillä käyttäjäsyötteillä. Koneen suorittamilla testeillä voidaan testata myös ohjelmiston toimintakykyä kovan kuormituksen alla syöttämällä ohjelmalle suuria määriä käyttötapauksia, joita ihmistestaaja ei yksinkertaisesti pystyisi syöttämään. Manuaaliset testit sen sijaan pyrkivät havaitsemaan käyttäjäkokemukseen vaikuttaviin ongelmiin, kuten ovatko sovelluksen painikkeet loogisissa paikoissa käyttäjälle tai onko sovelluksen käyttäminen muuten sujuvaa. (10)

2.4.2 Testiautomaatio

Nopeasti voisi kuvitella, että testiautomaatio sekä automaattinen testaaminen tarkoittavat samaa asiaa. Tämä ei kuitenkaan pidä täysin paikkaansa. Jatkuvan integraation perimmäisenä tarkoituksena on luoda testattuja ohjelmistokokoonpanoja jokaisesta koodimuutoksesta ohjelmiston koodeihin. Suuremmissa ohjelmistokokonaisuuksissa testattavien ohjelmistokomponenttien määrä on valtava, joten testit saattavat viedä erittäin suuren määrän aikaa ja aiheuttaa pullonkaulan julkaisuputkeen. Tämä saattaa ruuhkauttaa jatkuvan integraation työkalua ja hidastaa julkaistavien ohjelmistokokoonpanojen luontia suuremmissa organisaatioissa ja aiheuttaa viivästymisiä julkaisemiseen asiakkaille. Jos ohjelmiston kehittäjiä on monta, niin ohjelmiston lähdekoodeihinkin tulee luonnollisesti enemmän muutoksia ja tämän myötä myös suurempi määrä ohjelmistokoosteita luodaan jatkuvan integraation ympäristössä.

Testiautomaation tarkoituksena on tarkastella tehtyjä muutoksia koodiin ja selvittää automaattisesti muutosten vaikutus ohjelmistokokonaisuudessa. Muutokset saattavat mahdollisesti vaikuttaa vain hyvin pieneen osaan ohjelmistoa, joten suurin osa testeistä saattaa olla turhia, jos testit kattavat koko ohjelmiston jokaisella testikerralla. (11)

2.4.3 Testaustasot

Ohjelmistotestaus jaetaan tavallisesti useampaan eri osaan, joilla pyritään varmistamaan ohjelmiston laatu mahdollisimman tehokkaasti. Tasot on jaettu niin, että testaus aloitetaan aina ohjelmiston pienimmistä komponenteista ja siirrytään aina suurempiin kokonaisuuksiin tasojen edetessä. Tällä tavalla voidaan havaita pienimmät virheet ohjelmistossa mahdollisimman aikaisessa vaiheessa ja korjata nämä ennen kuin ohjelmistoa aloitetaan testaamaan suuremmassa kokonaisuudessa. (12, s. 50 - 51)

Yksikkötestaus

Yksikkötestit ovat ohjelmiston kaikista pienin osa, jossa pyritään testaamaan pieniä ja yksikertaisia toimintoja ohjelmistossa. Nämä testit tarkastelevat esim. ohjelmiston funktioissa kulkevia olevia arvoja, jotta arvot ovat hyväksyttäviä ja varmistavat ohjelman oikeanoppisen toiminnan. Yksikkötestaamisella voidaan myös välttää ohjelmistoa korruptoimasta tiedostoja, joita ohjelmisto käyttää ajon aikana. (12, s. 51 - 53).

Integrointitestaus

Kun yksikkötestaus on suoritettu ja läpäisty, niin tämän jälkeen komponentti tai komponentit voidaan siirtää osaksi viimeksi testattua ja testien läpäissyttä kokonaisuutta. Jos komponentti on jo aiemmin ollut tässä koosteessa, niin tämä komponentti korvataan uudemmalla komponentilla ja testataan muutoksen vaikutusta ohjelmistossa. Integrointitestauksessa testataan yleensä komponenttien toimintaa osana järjestelmää ja komponentin keskustelua muiden komponenttien kesken.

Integrointitestauksessa saatetaan käyttää myös sijaiskomponentteja, jotka eivät ole ohjelmistokoosteen ohjelmiston lopullisia komponentteja, mutta itse testattavan komponentin toimintaan vaadittavia osia. (12, s. 51 - 53)

Järjestelmätestaus

Ohjelmiston tai järjestelmän toiminta kokonaisuudessa testataan järjestelmätestaus vaiheessa, jossa ei enää käytetä sijaiskomponentteja. Järjestelmä on koottu ja siirretty tässä vaiheessa ympäristöön, joka vastaa ohjelmiston kohdeympäristöä julkaistavassa versiossa. Järjestelmätestauksen aikana testataan tämän hetkiselle versiolle ominaisia kohdevaatimuksia, jotka saattavat olla myös ohjelmiston käytettävyyteen liittyviä testitapauksia. Tässä vaiheessa testausta saatetaan myös tarvita ihmistestaajaa testaamaan ohjelmiston toimintaa, eli automatisoidut testitapaukset eivät välttämättä enää riitä. Testausvaiheen aikana voidaan kuitenkin suorittaa joitain testitapauksia, jotka ovat suoritettavissa automaattisesti ja vaativat lähes täydellisen ympäristön testitapausten testaamiseen. (12, s. 56 - 57)

Hyväksymistestaus

Hyväksymistestaus on testauksen viimeinen taso, jossa ohjelmiston toimintavaatimukset testataan virallisesti. Hyväksymistestauksessa ohjelmisto testataan sen omassa kohdeympäristössä niin, että asiakas voi hyväksyä ohjelmiston toiminnan niin, että ohjelmiston omistus voidaan siirtää asiakkaalle. Asiakkaan hyväksyttyä ohjelmiston ohjelmisto siirtyy pois kehitysvaiheesta jälkituotantoon. Tässä vaiheessa testausta ohjelmistosta ei enää etsitä komponentti kohtaisia virheitä vaan vaihe keskittyy ohjelmistokokonaisuuteen tai toimintavaatimusten todentamiseen. (12, s. 57)

2.5 Ohjelmistolisenssit

Lisenssienhallinta on erittäin tärkeä osa tuottoisaa liiketoimintaa. Valitettavasti tästä huolimatta lisenssiehtojen seuranta saattaa jäädä monella yrityksellä hieman taka-alalle, varsinkin jos työtehtävää ei osoiteta kenellekään yrityksen sisäisesti. Lisenssiehtojen välinpitämättömyyden tuloksena voi yritykselle koitua lisämenoja, omien lähdekoodien vaarantamista tai jopa yrityksen kaatuminen on mahdollista. Erityisesti vapaan lähdekoodin lisenssiehdot saattavat unohtua helposti, koska nämä saattavat tulla osana ilmaiseksi ladattavassa paketissa internetistä. (13)

2.5.1 Avoimen lähdekoodin lisenssit

Open Source Initiative (”OSI”) on avoimeen lähdekoodiin perehtynyt yhdistys, joka vastaa avoimen lähdekoodin sertifioinnista (14). Avoin lähdekoodin lisenssi saa ”open source” sertifioinnin, kun lisenssin ehdot vastaavat OSI:n määritelmiä. Nämä määritelmät takaavat lisenssin alla olevan lähdekoodin kopiointi-, levitys- sekä muunteluvapauden. (15, s. 189 - 190)

Pohjimmiltaan avoimen lähdekoodin tarkoitus on avoin tuotekehitys. Avoimen lähdekoodin lisensseihin on kehitetty lisenssiehtoja, joilla pyritään varmistamaan avoimen lähdekoodin vapaa jatkokehitys. Nämä lisenssiehdot eivät usein salli lähdekoodin kehittämistä avoimen lähdekoodin yhteisön ulkopuolella. Tämä mahdollistaa lähdekoodin avoimen jatkokehityksen.

2.5.2 Avoimen lähdekoodin lisenssityyppejä

Vastavuoroisuutta vaativa lisenssi

Kun avoin lähdekoodi julkaistaan vastavuoroisuutta vaativan lisenssin alla, niin tämä koodin käyttö vaatii uuden kirjoitetun koodin julkaisemista samoin ehdoin kuin lisenssin alla oleva koodi, jota käytettiin uuden koodin tai käännöksen luomisessa. Tyypistä käytetään myös nimitystä *Copyleft*. Tämä lisenssityyppi voidaan lisäksi jakaa alatyyppeihin, jotka määräävät koodin käyttöä vielä hieman tarkemmin:

Tarttuva vastavuoroisuutta vaativa lisenssi (”Viral license”) vaatii, että lähdekoodin käyttäminen vaatii, että tuotekokonaisuus lisensioidaan samalla lisenssillä, mitä avoin lähdekoodi käyttää. Tämä tunnetaan myös GNU:n GPL-lisenssinä. (15, s.191 - 192)

Pysyvä vastavuoroisuutta vaativa lisenssi (”share alike”) taas sallii lähdekoodin linkityksen osaksi tuotekokonaisuutta ilman lisensiointimääräystä. Linkityksellä tarkoitetaan, että lähdekodin käännöstä voidaan käyttää esimerkiksi kirjastona tuotekokonaisuudessa. (15, s.191 - 192)

Tunnetuimpia vastavuoroisuutta vaativia lisenssejä ovat GPL (”General Public License”) ja LGPL (”Lesser General Public License”).

Sallivat lisenssit

Avoimen lähdekoodin lisenssityyppiä kutsutaan sallivaksi lisenssiksi tai yliopistolisenssiksi, jos tämä ei rajoita lähdekoodin käyttöä, muokkausta tai uudelleenjakamista millään tavalla. Nämä lisenssit eivät siis kehota lisensioimaan uutta tuotekokonaisuutta samalla lisenssillä.

Tunnetuimmat sallivat lisenssit ovat MIT (”Massachusetts of Information Technology”), BSD (”Berkeley Software Distibution”) sekä Apache lisenssi.

2.5.3 Avoin lähdekoodi yrityselämässä

Avointa lähdekoodia ja ilmaisohjelmia käyttäessä myytävää ohjelmistoa kehittäessä on oltava erittäin varovainen. Myytävää ohjelmistoa kehittäessä tulisi välttää käyttämästä vastavuoroisuutta vaativia lisenssejä. Tarttuvia vastavuoroisuutta vaativia lisenssejä ei tulisi ikinä käyttää ohjelmistoa kehittäessä, koska nämä vaativat lähdekooditiedostojen lisensioinnin samalla lisenssillä. Jos tällä lisenssillä kehitettyä ohjelmaa pyritään myymään, niin ohjelman tulisi sisältää myös ohjelman lähdekooditiedostot. Lähtökohtaisesti yritykset kuitenkin haluavat pitää omat lähdekoodit suljettuna, jotta ohjelmiston myynti olisi mahdollisimman suurta eivätkä mahdolliset kilpailijat pystyisi hyödyntämään lähdekoodeja. Jotta yritystoimintaa vaarantavilta avoimen lähdekoodin tiedostoilta voitaisiin välttyä, niin yrityksen tulisi kehittää sisäisiä käyttösääntöjä näihin liittyen. Ensinnäkin avointa lähdekoodia ei tulisi ottaa käyttöön ilman lupaa tai tutkimatta lisenssiehtoja erittäin huolella.

Koodin tai ilmaisohjelman lisenssin tarkastelu saattaa kertoa, voiko koodia käyttää lisensioimatta omaa koodia myös samalla lisenssillä. Yliopistolisenssit ovat tarpeeksi sallivia lisenssejä ja voidaan täten hyväksyä saman tien. Vastavuoroisuutta vaativia lisenssejä taas tulisi tarkastella tapauskohtaisesti. Tarkastelussa tulisi ottaa huomioon lähdekoodin käyttötarkoitus sekä yhteys omaan koodiin. (15, s. 207)

3 Toteutusmenetelmät

3.1 Jatkuvan integraation työkalu

Tässä työssä käytetään Jenkins-nimistä ohjelmaa jatkuvan integraation työkaluna (myöh. CI-työkalu). Jenkins on vapaan lähdekoodin ilmaissovellus, ja täten myös mahdollisesti eniten käytetty CI-työkalu. Jenkins palvelin on yhteensopiva Linuxin ja Windowsin kanssa, ja mahdollistaa myös molempien käyttöjärjestelmien "keskustelemisen" keskenään. Tämä ominaisuus on erittäin tärkeässä osassa tässä, koska ohjelmistoja tullaan kääntämään sekä Windowsille että Linuxille.

Jenkinsiä käytetään lähinnä ohjelmien automatisoituun kääntämiseen sekä testaamistarkoituksiin, mutta tällä työkalulla voi käytännössä tehdä paljon muutakin. Ohjelmien käyttöönottoon saattaa liittyä paljon muutakin kuin pelkästään koodien kääntäminen, esimerkiksi tietokantojen päivitystä ja muiden tiedostojen kopioimista ulkoisista lähteistä ohjelmaa varten.

3.2 Testiautomaatiotyökalu

Robot Framework toimii tässä työssä työkaluna, joka hoitaa testiautomaation julkaisuputkessa. Robot Framework on avoimen lähdekoodin sovellus, jolla voidaan ajaa käyttäjän luomia skriptejä. Skriptit toimivat Robot Frameworkin luomalla avainsanakielellä, joka on erittäin helppokäyttöinen. Ohjelma tunnistaa monia erilaisia Robot Frameworkiin luotuja avainsanoja, joilla voidaan suorittaa yksinkertaisia testejä suorittamalla tietokoneen tavallisimpia komentoja, kuten tiedostovertailua. Lisäksi Robot Frameworkiin on ladattavissa laajennuksia, jotka tavallisesti sisältävät uusia avainsanoja skriptejä varten. Näillä laajennuksilla voidaan taas lisätä toiminnallisuusvaihtoehtoja käyttöjärjestelmän ulkopuolelle, kuten verkkosivujen toiminnallisuuksien testaamista selaimella.

3.3 Lisenssienhallinnan menetelmät

3.3.1 Tunnistus

Koska lisenssienhallintaa ei ole ennen suoritettu tällä tavalla yhtenäisenä operaationa CI-ympäristössä, niin lisenssientunnistukseen on nähty parhaaksi käyttää Linuxin omia shell-skriptejä. Shell-skriptit ovat erittäin tehokkaita työkaluja tällä tasolla työskentelyyn, koska niitä voidaan ajaa suoraan Jenkinsistä niin, että skriptin tulosteet näkyvät myös Jenkinsissä. Lisäksi skriptien kehittäminen ja ongelmien korjaaminen helpompaa, kuin mitä omalla ohjelmalla olisi. Jos hallintaa varten kehitettäisiin oma ohjelmisto, niin luultavasti täytyisi ladata myös monia muita koodikirjastoja, jotta ohjelmalla päästäisiin läheskään sitä tasoa, mitä skripteillä voidaan saada aikaan. Lisenssitietojen tallennuksessa käytetään tietokantoja, joita voidaan taas kutsua suoraan skripteistä.

Ack

Työssä käytetään lähdekooditiedostojen hakemiseen ja tiedostojen erittelyssä paljon Linux-pohjaista Ack-ohjelmaa. Ack on vaihtoehto Linuxin perinteiselle grep -komennolle, ja soveltuu tähän työhön paljon paremmin kuin grep-komento. Ackin tehokkuus perustuu mahdollisuuteen käydä pelkästään läpi tiettyjä tiedostomuotoja jostain polusta. Näin ollen haku voidaan siis rajata pelkästään lähdekooditiedostoihin, esim. C++:n headerit ja .cpp-tiedostot.

3.3.2 Tietojen tallennus

Tietojen tallennusmenetelmänä käytetään MariaDB:tä. MariaDB on tietokantajärjestelmä, joka perustuu erittäin tunnetun MySQL:n koodeihin. Tietokantoja käytetään tässä työssä lisenssitietojen tallennukseen, siten että käyttöliittymä ja CI-ympäristö voivat molemmat käyttää tätä tietokantaa ja keskustella tämän kautta keskenään. MariaDB sekä käyttöliittymän PHP muodostavat yhdessä lisenssienhallinnan backendin.

3.3.3 Käyttöliittymä

Käyttöliittymää varten tehdään verkkosivu Jenkinsin kanssa samaan lähiverkkoon. Käyttämällä tavallisimpia verkkosivujen kehitykseen liittyviä kieliä (CSS, HTML, JavaScript, PHP) voidaan saada yksinkertainen, mutta toiminallisuudeltaan laaja käyttöliittymä. Käyttöliittymän toimintaan kuuluu lisenssitietojen hakeminen MariaDB:n tietokannoista PHP:n avulla. Lisenssitiedostot esitetään hakemisen jälkeen käyttäen taas hyväksi HTML, CSS-koodia. Javascript toimii käyttöliittymässä tämän toimintoja ohjaavana koodikielenä. HTML, CSS sekä Javascript toimivat siis lisenssienhallinan frontendinä.

4 Toteutus

4.1 Jatkuva integraatio

Jatkuvan integraation toteutus sisältään paljon informaatiota, joka saattaa olla toimeksiantajalle yksityistä. Näiden toteutusratkaisujen esittäminen ei välttämättä käy edes järkeen ulkopuoliselle tarkastelijalle, joten tämän kappaleen tarkoituksena on pikemminkin esitellä näitä toteutusratkaisuja ja muutoksia yleisellä tasolla. CI-ympäristön päivittäminen toi mukanaan paljon uusia ominaisuuksia, jotka voidaan kuitenkin esitellä. Näiden ominaisuuksien käyttöä pyritään esittelemään mahdollisimman perustellusti, jotta näiden ominaisuuksien käyttötarkoitus itse työssä tulisi selväksi lukijalle.

4.1.1 Jenkinsin käsitteet

Jobit

Jobit ovat Jenkinsissä suoritettavia työtehtäviä, joita voidaan suorittaa ajastetusti tai manuaalisesti. Käyttäjälle jobi toimii eräänlaisena skriptinä, jolla voidaan suorittaa palvelimella sarjassa eri operaatioita. CI:ssä nämä operaatiot saattavat olla esimerkiksi kääntäjän kutsuminen jollekin käännettävälle ohjelmalle. Jobiin liittyy myös Jenkinsin puolella erilaisia asetuksia, jolla voidaan saada jobille tietynlaisia ominaisuuksia, kuten ajastus jobin suorittamiselle tai erilaisia parametrejä, joita Jenkins käyttää jobin suorituksen aikana.

Pipeline

Tunnetaan myös Jenkinsin vanhemmissa versioissa nimellä Workflow. Pipelinen tehtävänä on jaotella CI:n prosessiketju eri tasoihin. Tasoilla pystytään luomaan ohjelmiston rakennukselle looginen järjestys, jotta ohjelmiston toiminta voidaan varmistaa kääntämällä koodit oikeassa järjestyksessä sekä kokoamalla resurssit oikein.

Slave

Jenkinsin slavella eli orjalla tarkoitetaan Jenkins palvelimeen yhteydessä olevaa toista konetta, jolle Jenkins voi tarvittaessa delegoida jobien suorituksia. Tämä on erittäin käytännöllinen sekä oleellinen osa monipuolisen CI-ympäristön kehityksessä. Orjat voivat CI:ssä olla esimerkiksi testikoneita, joille ohjelmistot voidaan lähettää sekä ajaa. Tällä voidaan lisäksi testata ohjelmiston laitteistoriippumattomuutta erittäin yksinkertaisesti. Orjat voivat olla siis eri käyttöjärjestelmiä, joille ohjelmiston käännösten tekeminen voidaan delegoida. Orjien käyttöönotto vaatii sen, että nämä laitteistoille asennetaan kaikki tarvittavat ohjelmistokomponentit, joilla käännökset voidaan kääntää tai testata.

4.1.2 Jenkins 2.0

Eräs suurimmista muutoksista työhön liittyen oli Jenkinsin päivittäminen uudempaan versioon. Päivitetty Jenkins käyttää 2.X versiota, joka sisältää paljon uusia parannuksia aiempiin Jenkins versioihin verrattuna. Tämän keskeisin muutos on kuitenkin uudistettu Pipeline.

2.0:aa edeltävissä versioissa workflow on toteutettu ajamalla Jenkinsin jobeja sarjassa. Sarja määräytyi käyttäjän määrittämistä ylävirta sekä alavirtajobeista. Näiden virtojen tarkoituksena on kuvata jobien suoritusjärjestyksiä toisiinsa verrattuna. Jobilla on tavallisesti vain yksi ylävirtajobi, mutta saattaa olla useampi alavirtajobi. Virta aloitetaan yhdestä jobista, jonka suorituksen jälkeen kaikki tämän jobin alavirtajobit käynnistetään palvelimen omissa säikeissään. Tätä virtaa jatketaan niin kauan, että workflow pääsee vaiheeseen, että yhdelläkään jobilla ei ole enää suoritettavia alavirtajobeja.

Jenkins 2.0 taas otti taas uuden pipeline mallin käyttöön, korjaten samalla muutamia ongelmia liittyen workflowiin. Workflow mallinnus on hidas, johtuen lukuisista jobien hallintaan liittyvistä viestintäviiveistä palvelimella. Lisäksi workflow:n graafinen näkymä Jenkinsissä ei ole kovin toimiva. Jobien suhteiden kuvaus oli näkyvissä eräänlaisessa kaaviossa, joka on hyvin sekavan näköinen.

Jenkins 2.0:ssa pipelinelle on oma jobi, jossa voidaan yksinkertaisesti suorittaa samat operaatiot, jotka edeltävässä versiossa suoritettiin useammassa jobissa. Tälle jobille rakennettiin oma skriptirakenne, jotta käyttäjä voi itse koodata oman pipelinensä käyttäen

Groovy-skriptikieltä sekä Jenkinsin omia funktioita. Lisäksi jobille rakennettiin oma graafinen ulkoasu, jossa pipelinen suoritushistorian näkee tasoittain. Kuvassa 5 on esimerkki Jenkins 2.0:n tasonäkymästä viidellä eri suorituskerralla eri tuloksin.

Kuva 5. Pipelinen tasonäkymä Jenkins 2.0:ssa

4.1.3 Jenkinsfile

Pipelinen skripti on mahdollista koodata Jenkinsin kautta jobiin. Tällä skriptillä käyttäjä voi varmistaa sen, että käännettävän ohjelmiston kaikki komponentit käännetään varmasti oikeassa järjestyksessä, eivätkä käännösten riippuvuussuhteet vaikuta pipelinen kulkuun. Ohjelmistokehitys on kuitenkin jatkuva prosessi ja ohjelmistokokonaisuus saattaa ajan mittaan muuttua; käännettäviä komponentteja saattaa tulla lisää, vanhoja poistua sekä pipelinen sisäiset operaatiot saattavat muuttua ohjelmiston versionumeroiden myötä. Jos skripti koodataan jobiin niin tämä tarkoittaisi sitä, että jokaiselle ohjelmiston versionumerolle pitäisi tehdä oma jobi kääntämistä varten. Tätä ongelmaa varten kuitenkin kehitettiin Jenkinsfile, joka voidaan siirtää versionhallintaan. Tämä tiedosto sisältää versiohallinnassa olevalle versionumerolle ominaisen skriptin, jonka Jenkins voi lukea jobin prosessoinnin alussa. Tämän suoritus menetelmän hyöty on myös, että kehittäjät voivat itse helposti lisätä omia komponenttejaan jenkinsfileen komponenttien luomisen yhteydessä.

4.1.4 Parametrisointi

Jenkins 2.0 on mahdollistanut monia uusia toiminallisuuksia CI-ympäristössä. Näillä toiminallisuuksilla voidaan pyrkiä saamaan ohjelmistokoosteen tekemisestä mahdollisimman helppoa, vaikka todellisuudessa tämä olisikin erittäin monivaiheinen ja työläs operaatio manuaalisesti tehtynä. Koosteen tekemiseen saattaa liittyä erilaisia komplikaatioita liittyen esimerkiksi resursseihin, joita ohjelma tarvitsee toimiakseen testiautomaatio prosessin aikana. Jenkinsfile kappaleen aikana kerrottiin, miten nämä resurssit saattavat myös olla ohjelmiston eri versionumeroille ominaisia muuttujia. Parametrisoimalla ohjelmistoresurssit voidaan välttää resurssien kovakoodaaminen Jenkinsfileen. Jenkinsistä löytyy parametrisointia varten ratkaisu, jossa käyttäjä voi itse määrittää omat parametrit omaa jobiaan varten. Parametreillä voidaan vaikuttaa pipelineen hyvin monipuolisesti. Työssä käytetään parametrisointia muun muassa pipelinen tasojen suorituksen määrittelemiseen. Pipeline sisältää muutamia tasoja, jotka vievät paljon aikaa. Nämä tasot ovat kuitenkin sellaisia, joita ei välttämättä tarvitsisi suorittaa ohjelmiston jokaisen iteraation aikana, mutta kuitenkin välttämättömiä suorittaa tasaisin väliajoin. Tasoille voidaan määritellä omat parametrit, joilla voidaan ohittaa nämä aikaa vievät tasot nopeimmissa käännöksissä. Parametrit voidaan asettaa oletusarvoisesti ohittamaan nämä tasot ja pitämään käännösajat lyhyinä. Lisäksi tällä ratkaisulla voidaan antaa myös käyttäjille mahdollisuus vaikuttaa tasojen suorituksiin, mikäli käännöksiä tehdään manuaalisesti. Kuvassa 6 on Parametrienvalintaruutu, kun pipelinen suoritus käynnistetään manuaalisesti.

Kuva 6. Parametrinäkymä Jenkisissä, kun iteraatio käynnistetään manuaalisesti

Parametrinä voidaan käyttää jobissa esimerkiksi bool-muuttujaa, joka on esitetty Jenkinsissä yksinkertaisena valintaruutuna. Jobin asetuksista voidaan näille määrittää näille parametreille oletusarvot, joita käytetään aina, kun käännös aloitetaan automatisoidusti.

Kun parametrisointi on saatu kuntoon jobissa, niin sitä voidaan käyttää seuraavasti Jenkinsfilessä:

```
stage(”Stage 1”)
{
	if(env.parameter =! true)
	{
		echo ”Ohitetaan tämä taso…”
	}

	else
	{
	//	Suoritetaan tason toimenpiteet
	}
}
```

4.1.5 Orjien käyttäminen Jenkinsfilessä

Tavallisesti Jenkins jakaa jobit itse. Jos Jenkinsiin on asennettu useita orjia, jobin koosteen tekeminen delegoidaan jollekin vapaana olevalle orjalle. Jos Jenkins palvelimen suorittimet ovat vapaana silloin kun kooste käynnistetään, niin Jenkins saattaa tehdä tämän myös itse. Jobien asetuksista voidaan määrittä jobeille oma vakio-orja, joka hoitaa aina tämän kyseisen jobin.

Joskus kuitenkin saattaa olla, että jokin jobi vaatii enemmän kuin yhden orjan samassa jobissa, kuten esimerkiksi testikoneille lähettäminen. Tämä voidaan hoitaa Jenkinsfilessä asettamalla suoritettava tehtävä tai tehtäväsarja *node*-nimisen funktion sisään:

```
stage(’Taso’)
{
	node(’Orja 1’)
	{
		//Prosessit Orja 1:llä
	}
	node(’Orja 2’)
	{
		//Prosessit Orja 2:lla
	}
}
```

Tämä menetelmä on hyvin yksinkertainen, mutta tehokas tapa suorittaa prosesseja useita eri orjia käyttämällä. Tällä menetelmällä voidaan siis esimerkiksi antaa orjille käsky hakea Jenkins-palvelimelta viimeisin kooste ohjelmistosta sekä ajaa se käyttämällä orjakoneen shell-komentoja.

Tässä työssä kuitenkin tämä menetelmä ei aivan riitä. Edellä mainittu on kovakoodattu menetelmä, jossa prosessit suoritettaisiin aina samoja orjia käyttämällä. Työssä käytetään parametria, jolla voidaan itse valita käytettävät orjat. Tämä ratkaisu ei tue kunnolla tätä ominaisuutta. Ja koska käytettävien orjien määrä saattaa vaihdella, niin *node*-funktion täytyy saada silmukan sisään. Silmukalla voidaan varmistaa, että kaikki valitut prosessit tullaan suorittamaan valituilla orjilla niiden lukumäärästä huolimatta:

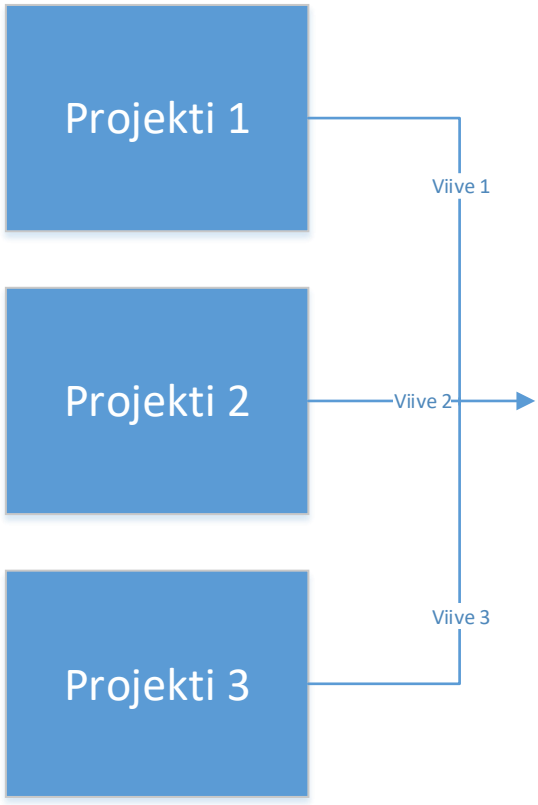
```
def labels = str.split(',')
def builders = [:]
for(int i = 0; i < labels.size(); i++)
{
        String label = labels[i]
        builders["builder${i}"] =
        {
            node(label)
            {
               //Prosessit i:n osoittamalla orjalla
            }
        }
    }
    parallel builders
}
```

4.1.6 Rinnakkaiset suoritukset

Suurien ohjelmistokokonaisuuksien kääntäminen saattaa viedä paljon aikaa riippuen ohjelmiston koosta. Ohjelmisto saattaa lisäksi koostua monista eri projekteista, jotka täytyy kääntää kukin erikseen. Ajan käyttöä lisäävät projektien käännösten väliset viiveet, jotka muodostuvat deallokointiajoista muistissa, kommunikointiviiveestä CI-työkalun ja kääntäjän välillä sekä kääntäjän valmistelusta seuraavaan projektiin, mikäli käännökset suoritetaan putkessa. Kuvassa 7 on esimerkki viiveistä, jotka sijoittuvat projektien väleihin.

Kuva 7. Putkessa kääntämisen havainnollistus.

Putkessa kääntämisen sijaan Jenkins 2.0 mahdollistaa projektien kääntämisen rinnakkain. Rinnan kääntäminen vähentää viiveaikaa huomattavasti, koska kääntäminen voidaan jakaa prosessorin useammalle säikeelle samaan aikaan ja viesti CI-työkalulta kulkeutuu näille säikeille samanaikaisesti. Kuvassa 8 on havainnollistettu viiveiden kesto rinnakkaisissa suorituksissa.



Kuva 8. Rinnakkain kääntämisen havainnollistus

Rinnan suorittaminen hyödyntää paremmin myös koneen suoritustehoa käyttämällä paremmin hyödyksi prosessorin useampaa ydintä.

Jenkinsfilessä rinnakkaisia suorituksia voidaan kutsua *parallel*-avainsanalla. Tämän avainsanan jälkeen kaikki tehtävät toiminnot tai käännökset voidaan kirjoittaa sulkujen sisään. Kaikki sulkujen sisässä olevat toiminnot täytyy nimetä ja erotella pilkuilla:

```
parallel
(
      Projekti 1:
      {
            //Toiminnot projekti 1:lle
      },
      Projekti 2:
      {
            //Toiminnot projekti 2:lle
      },
      Projekti 3:
      {
            //Toiminnot projekti 3:lle
      }
);
```

4.1.7 Jenkinsin artefaktit

Kun yrityksellä on käytössä täysin toimiva Jatkuvan Integraation ratkaisu, niin kehitettävästä tuotteesta käännetään mahdollisesti kymmeniä ohjelmistokokonaisuuksia päivässä. Jokainen käännöksessä on kuitenkin pieniä eroja, joita ohjelmiston kehittäjät ovat tehneet koodiin ennen kuin he ovat lähettäneet koodimuutokset versionhallintapalvelimelle. Välillä kuitenkin nämä muutokset eivät toimi ohjelmistokokonaisuudessa, vaikka kehittäjä olisikin testannut omaa koodiansa pienemmässä koosteessa omalla koneellaan. Tämä saattaa aiheuttaa CI-palvelimella sen, että epäonnistuneet tai vialliset käännökset ylikirjoittavat toimivat ohjelmistokokonaisuudet CI-palvelimelta. Tästä syystä on hyvä tehdä jokaisesta onnistuneesta iteraatiosta oma artefakti CI-palvelimelle. Jenkinsissä on oma *archiveArtifacts*-avainsana, jolla voidaan arkistoida jokin kohdetiedosto tai kansio CI-palvelimelle. Ideaalissa tilanteessa tämä arkistoitu tiedosto olisi ohjelmistokoosteen pakattu versio, joka sisältää kaikki tarvittavat komponentit ohjelmiston toimimiseksi. Näin ollen arkistoidut toimivat ohjelmistot ovat helposti saatavilla, mikäli näitä halutaan käyttää.

Artefaktit voidaan arkistoida seuraavasti Jenkisfilessä:

```
archiveArtifacts"Tiedostopolku"
```

4.2 Lisenssitietokanta

Lisenssitietokanta on käyttöliittymän, lisenssientunnistusskriptin sekä myöhemmin julkaisuputkessa oleva lisenssitiedoston generoivan skriptin yhteinen tiedon tallennuspaikka.

Lisenssitietokannan taulut on rakennettu siten, että sinne voidaan tallentaa tiedot lähdekooditiedostoista, tunnetuista avoimen lähdekoodin lisensseistä sekä ryhmitetyistä lähdekooditiedostoista.

4.2.1 Lähdekooditiedostojen listat

Listat ovat lähdekooditiedostojen jakomenetelmä. Listoilla pidetään kirjaa kaikista lähdekooditiedostoista, jotka sisältävät tekijänoikeuden ilmaisun ja vaativat jonkinlaisia toimenpiteitä. Nämä listat on jaettu seuraavasti; verifioimattomat, verifioidut sekä musta lista.

Jokaisen listan taulut on kirjoitettu samalla tavalla. Taulujen kentät ovat kirjoitettu lähdekooditiedostojen kannalta vaativien tietojen mukaisesti. Näitä tietoja ovat identiteettinumero, tiedoston nimi, tiedostopolku versionhallinnassa sekä tiedostopolku CI-palvelimen koneen juuresta. Jotta tietokannassa vältyttäisiin duplikaateilta, identiteettinumerosta ja versionhallinnan tiedostopolusta täytyy tehdä uniikkeja kenttiä. Lisäksi identiteettinumerosta voidaan tehdä auto-inkrementaalinen kenttä, joka hoitaa identiteettinumeroinnin automaattisesti:

```
Create Table 'listannimi'
(
      id INT PRIMARY KEY AUTO_INCREMENT NOT NULL
      filename VARCHAR(50),
      initial_path VARCHAR(50),
      repository_path VARCHAR(50),
      UNIQUE(repository_path),
);
```

Verifioimattomat

Lähdekooditiedostot, jotka löytyvät pelkällä ”Copyright”-hakusanalla, lisätään ensimmäisenä verifioimattomien tiedostojen listalle. Tämä verifioimattomien lista toimii ikään kuin oletuslistana tunnistuksen aikana löydetyille tiedostoille. Verifioimattomaksi jääneitä tiedostoja voidaan siirtää käyttöliittymän kautta toisiin listoihin. Ideaalitilanteessa verifioimattomien listalla ei pitäisi olla yhtään tiedostoa, vaan kaikki tiedostot olisi jaettu joko verifioitujen listalle tai mustalle listalle.

Verifioidut

Verifioitujen listalle siirretään kaikki tiedostot, jotka on tarkastettu käyttöliittymän kautta ristiriidassa olevien lisenssiehtojen varalta käännettävän ohjelmiston kanssa. Tämän listan kautta lähdekooditiedostot voidaan jakaa lisenssiryhmiin, joista generoidaan lisenssitekstitiedosto myöhemmässä vaiheessa julkaisuputkea.

Musta lista

Lista lähdekooditiedostoille, joiden lisenssiehdot eivät mahdollisesti ole kehitettävän ohjelmiston käyttötarkoituksen kanssa yhteensopivia. Nämä tiedostot ovat tarkkailun alla ja vaativat mahdollisesti toimenpiteitä. Lähdekooditiedostojen käyttötarkoitus tulisi tämän jälkeen tarkastaa, että käännetäänkö näistä lähdekooditiedostoista mahdollisesti ohjelmiston toimimiseen vaadittavia komponentteja. Mikäli komponentit eivät ole osana ohjelmistoa, tai jos lähdekooditiedostoa ei käytetä kääntämiseen ollenkaan, tiedosto voidaan lisätä verifioitujen listalle. Jotkut avoimen lähdekoodin lisenssit saattavat sallia lähdekoodin käännöksen dynaamisena kirjastona, joten myös tämä täytyy ottaa huomioon mustaa listaa läpi käydessä.

4.2.2 Avainsanat

Avainsanat ovat käyttöliittymän kautta manuaalisesti syötettyjä sanoja tai lauseita, joita voidaan hakea lähdekooditiedostoista. Nämä ovat ”Copyright”-parametrin jälkeen heti seuraava haku parametri, joilla voidaan helposti vähentää tutkittavien lähdekooditiedostojen määrää. Avainsanoilla voidaan helposti karsia esimerkiksi yrityksen omia lähdekooditiedostoja, joissa noudatetaan jotain vakiintunutta tekijänoikeuden ilmaisua. Avainsanat on jaettu kahteen tyyppiin; mustiin sekä valkoisiin avainsanoihin.

Mustat avainsanat ovat sanoja tai lauseita, joita halutaan varoa lähdekooditiedostoissa. Kun näitä avainsanoja löytyy lähdekooditiedostoista, niin nämä tiedostot siirretään automaattisesti lähdekooditiedostojen mustalle listalle. Mustalle listalle voi esimerkiksi lisätä vastavuoroisuutta vaativiin lisensseihin viittaavia sanoja, kuten GPL ja LGPL.

Valkoiset avainsanat ovat vastaavasti taas sanoja, jotka voidaan päästää läpi tarkastuksesta. Nämä sanat poistetaan lisenssien etsimisvaiheen aikana listalta eikä käyttäjän tarvitse tarkastaa näitä. Hyviä valkoisia avainsanoja ovat esimerkiksi yrityksen omien lähdekooditiedostojen tekijänoikeuden ilmaisuja.

Avainsanojen taulut on luotu MariaDB:ssä seuraavasti:

```
Create Table 'Avainsanan tyyppi'
(
      id INT PRIMARY KEY AUTO_INCREMENT NOT NULL
      name VARCHAR(50),
      UNIQUE(name)
);
```

Taulussa *name* vastaa avainsanaa ja *id* identiteettinumeroa. Nimi ja identiteettinumero ovat molemmat uniikkeja kenttiä, eikä näissä kummassakaan voi olla duplikaatteja.

4.2.3 Avoimen lähdekoodin lisenssien lista

Avoimen lähdekoodin lisenssien lista on mahdollisesti kaikista tärkein lista lisenssien tunnistusprosessissa. Tämä lista on myös jaettu valkoiseen ja mustaan listaan, samalla tavalla kuin avainsanat. Näitä listoja voidaan käyttää suoraan verrannollisesti sallivien lisenssien ja vuorovaikutusta vaativien lisenssien kanssa, eli sallivat lisenssit lisätään valkoiseen listaan ja vuorovaikutusta vaativat mustaan listaan. Näihin listoihin lisättyjä lisenssimalleja kutsutaan tämän työn yhteydessä lisenssitemplaateiksi.

Lisenssitemplaattitaulu pitää sisällään neljä kenttää; identiteettinumeron, templaatin nimen sekä lisenssiehdot määrittelevän tekstin. Identiteettinumero ja templaatin nimi ovat uniikkeja kenttiä:

```
Create Table 'Lisenssilistan tyyppi'
(
      id INT PRIMARY KEY AUTO_INCREMENT NOT NULL
      name VARCHAR(50),
      notice TEXT,
      UNIQUE(name)
);
```

4.2.4 Tiedostopolkujen sivutukset

Versionhallinnassa saattaa olla myös tiedostopolkuja, joihin laitetaan kaikki käyttämätön koodi. Koodit saattavat sisältää esimerkiksi mahdollisia mustan listan avoimen lähdekoodin lisenssejä, joita on mahdollisesti käytetty prototyyppejä varten, eivätkä kuulu siis kehitettävän ohjelmiston kokoonpanoon. Näitä tiedostopolkuja varten on luotu oma taulu, joka ei tarvitse kuin kaksi kenttää:

```
Create Table ignore_folder
(
      id INT PRIMARY KEY AUTO_INCREMENT NOT NULL
      name VARCHAR(50),
      UNIQUE(name)
);
```

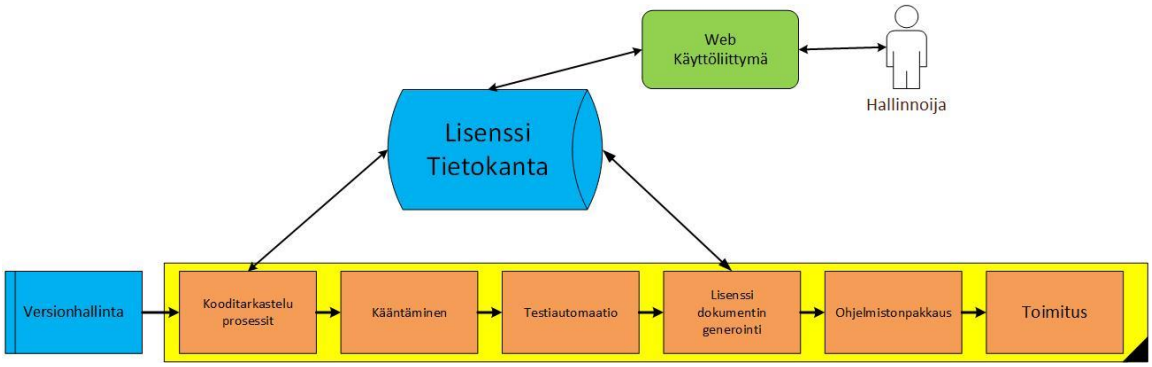
4.2.5 Lisenssiryhmät

Käyttöliittymän kautta verifioiduista lähdekooditiedostoista voidaan luoda omia lisenssiryhmiä. Lisenssiryhmillä voidaan siis koota kaikki lähdekooditiedostot yhteen, jotka kuuluvat samaan avoimen lähdekoodin ohjelmistoon. Tietokannan taulu on tehty lähes samalla tavalla kuin verifioitujen lista. Poikkeuksena on yksi lisäkenttä, jossa määritellään lisenssiryhmä *affliation*:

```
Create Table 'listannimi'
(
      id INT PRIMARY KEY AUTO_INCREMENT NOT NULL
      filename VARCHAR(50),
      initial_path VARCHAR(50),
      repository_path VARCHAR(50),
      affliation VARCHAR(50),
      UNIQUE(repository_path),
);
```

4.3 Avoimen lähdekoodin lisenssienhallinta osaksi julkaisuputkea

Jotta lisenssienhallinta voidaan liittää toimivasti CI-ympäristöön, niin osan siitä täytyy toimia manuaalisesti. Pipeline halutaan kuitenkin pitää täysin automaattisena, joten lisenssienhallinnan täytyy siis osittain haarautua pois julkaisuputken prosessiketjusta. Kuvassa 9 on esitetty lisenssienhallinnan haarautuminen julkaisuputken vaiheista.



Kuva 9. Lisenssienhallinnan haarautuminen julkaisuputkesta

Haarautuminen tapahtuu CI-putken alkuvaiheessa. Kun uusimmat koodit on haettu versionhallinnasta, niin CI-palvelimella voidaan tämän jälkeen tarkastella koodeja paikallisesti. Tarkasteluprosessissa aloitetaan lisenssien tunnistusoperaatio, jossa käydään koodit läpi avoimen lähdekoodin varalta. Tunnistuksen aikana tietoja syötetään tietokantaan, jonka tietoja taas voidaan käyttää suoraan käyttöliittymässä.

Käyttöliittymän tarkoituksena on näyttää lisenssien hallinnoijalle kooditarkastelu prosessien tulokset. Tulokset ovat interaktiivisessa muodossa, jotta hallinnoija voi tarkastella kooditiedostojen lisenssiehtoja ja jaotella kooditiedostoja näiden käyttötarkoituksen mukaan. Käyttöliittymän kautta lisenssitiedot saadaan käsittelyn jälkeen muotoon, josta voidaan generoida ohjelmistolle lisenssitekstitiedosto. Lisenssitekstitiedosto toimii kirjallisena ilmaisuna käyttäjälle käytetyistä avoimen lähdekoodin ohjelmista ja tiedostoista.

4.3.1 Lisensioitujen lähdekooditiedostojen hakeminen

Avoimen lähdekoodin tiedostojen tunnistaminen ei ole helppoa, sillä tekijänoikeus ei vaadi minkäänlaista merkintää lähdekoodiin tekijänoikeuden suojaamiseksi. Tekijänoikeuden ilmaiseminen kuitenkin helpottaa tekijää, lähdekoodin mahdollista jatkokehittäjää sekä lähdekoodia käyttävää kehittäjää mahdollisissa ongelmatilanteissa. (16)

Nykypäivänä lähdekooditiedostoista löytyy tavallisesti vähintään seuraavanlainen tekijänoikeuden ilmaisu:

```
Copyright © <Vuodet> <Tekijänoikeuden haltija>
```

Koska selkein ilmaisu lähdekoodissa on mainita ”Copyright” lähdekooditiedostossa, niin tätä voidaan käyttää erinomaisena hakusanana lähdekooditiedostojen hakemisessa.

Käyttäen ack:ia, komento *bash*-skriptissä näyttää seuraavalta:

```
ack -l -i --flush --cpp "Copyright"
```

Komento tulostaa polusta vuorotellen kaikki C++ -lähdekooditiedostot, joissa on *Copyright*-ilmaisu. Nämä tulosteet on hyvä kirjoittaa seuraavia komentoja varten johonkin tiedostoon kirjoittamalla *'> "tekstitiedoston nimi"* komennon perään.

4.3.2 Tunnettujen avoimen lähdekoodin lisenssien tunnistus

Kun Copyright–ilmaisun omaavat lähdekooditiedostot on saatu listattua tekstitiedostoon, niin lähdekooditiedostoille voidaan aloittaa tekemään tarkempaa tunnistusprosessia. Tämän prosessin kuvaamisen helpottamiseksi käytetään jatkossa esimerkkinä seuraavaa MIT–lisenssiä, joka löytyy monesti tällä lisenssillä lisensioiduista lähdekooditiedostoista:

```
Copyright © <Vuodet> <Tekijänoikeuden haltija>

Permission is hereby granted, free of charge, to any person obtaining a
copy of this software and associated documentation files (the "Software"),
to deal in the Software without restriction, including without limitation
the rights to use, copy, modify, merge, publish, distribute, sublicense,
and/or sell copies of the Software, and to permit persons to whom the
Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included
in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABIL-
ITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT
SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES
OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
OTHER DEALINGS IN THE SOFTWARE.
```

Tätä lisenssitekstiä voidaan käyttää mallina, kun lähdetään etsimään lähdekoodi tiedostoista MIT–lisenssejä. Teoriassa yksinkertaisin tapa olisi lähteä vertaamaan lähdekooditiedostojen ja lisenssimallin rivejä keskenään. Käytännössä tämä ei kuitenkaan onnistu, koska lisenssitekstit eivät välttämättä ole rivitetty lähdekooditiedostoissa samalla tavalla

kuin mallissa. Sana kerrallaan vertaaminen taas hidastaa tunnistusprosessia liikaa suurissa projekteissa, joissa saattaa olla mahdollisesti tuhansia lähdekooditiedostoja. Tässä työssä etsitään siis mallin rivin ensimmäistä ja viimeistä sanaa lähdekooditiedostosta. Tämä menetelmä ei ole riippuvainen lähdekooditiedostojen rivityksestä ja toimii paljon nopeammin kuin sana kerrallaan. Todennäköisyys virheellisille täsmäyksille on erittäin pieni riippuen lisenssitekstin rivimäärästä.

Kun avoimen lähdekoodin lisenssejä tunnistetaan skriptissä, niin ensimmäisenä tarvitaan tietokannasta avoimen lähdekoodin templaatin teksti. Tämän mallin lisenssiteksti on hyvä kirjoittaa ylös johonkin väliaikaistiedostoon, sillä se helpottaa huomattavasti lisenssitekstin vertailemista lähdekooditiedostojen lisenssiteksteihin skriptissä suoritettavissa komennoissa:

```
echo -e $(mysql -uKäyttäjä -pSalasana -hlocalhost -DLisenssitietokanta -s -N -e "SELECT notice FROM lisenssilista WHERE name='Lisenssinnimi'") > "Väliaikaistiedosto"
```

Tämä komento voidaan asettaa silmukan alle, jossa haetaan kaikki tietokantaan tallennetut lisenssimallit, ja suoritetaan näiden lukumäärän verran:

```
while read license; do

//Lisenssimallin operaatiot

done < <(echo "SELECT name FROM opensource_template_blacklisted " | mysql -uKäyttäjä -pSalasana -hlocalhost -DLisenssitietokanta | sed -n '1!p')
```

Nyt kun lisenssiteksti on kirjoitettuna väliaikaistiedostoon, niin tätä tekstiä voidaan ruveta vertailemaan lähdekooditiedostoihin. Vertailu suoritetaan käyttämällä silmukkaa, joka toistetaan lisenssimallin rivimäärän mukaan. Tässä komentosarjassa tarvitaan mukana kahta väliaikaistiedostoa. Ensimmäinen tiedosto (*Tiedosto1*) pitää sisällään listan kaikista tiedostoista, jotka tullaan tarkistamaan. Toinen tiedosto (*Tiedosto2*) taas on väliaikaistiedosto, johon listataan kaikki tarkistuksen kriteerit täyttäneet tiedostot. Jokaisen silmukan iteraation jälkeen Tiedosto1 korvataan Tiedosto2:lla, jotta silmukan päätteeksi Tiedosto1:stä löytyy vain kaikki kriteerin täyttäneet tiedostot eli toisin sanoen kyseisen lisenssin alla olevat tiedostot:

```
while IFS='' read -r line
do
	if [ "$line" ]; then

	words=( $line )
	LINE_START=$(echo "${words[0]}")
	LINE_END=$(echo "${words[-1]}")

	ack "${LINE_START}" -Q -l -i --files-from="Tiedosto1" | ack
	"${LINE_END}" -Q -x -l -i > "Tiedosto2"

	fi
done < "Väliaikaistiedosto"
```

Tämän jälkeen Tiedosto1:n lähdekooditiedostojen lista voidaan lisätä joko mustaan listaan tai verifioitujen listaan, riippuen siitä, käytiinkö silmukan aikana läpi mustan vai valkoisen listan lisenssimallia.

4.3.3 Avainsanojen tunnistus

Avainsanojen hakeminen lähdekooditiedostoista on yksinkertaista. Tähän ei tarvita kuin hakea tietokannasta kyseinen avainsana sekä antaa ack:lle tiedostoon kirjoitettu lista lähdekooditiedostoista, jotka tarkastetaan. Komennolla *files-from* voidaan antaa tekstitiedosto, johon lähdekooditiedostot on listattu riveittäin.

Ensin käytetään silmukkaa apuna, jotta voidaan käydä kaikki tallennetut avainsanat läpi. Linuxin *while*-silmukkaan voidaan antaa myös suoraan ulostulona lista avainsanoista:

```
while read i; do

//Silmukassa suoritettavat operaatiot

done < <(echo "SELECT name FROM ( avainsanan tyyppi)" | mysql -uKäyttäjä
-pSalasana -dLisenssitietokanta | sed -n '1!p')
```

Silmukan sisässä voidaan käyttää suoraan muuttujaa *i* suoraan ack:ssa hakusanana. Eli tämän jälkeen tarvitaan vain lista lähdekooditiedostoista (*Tiedosto1*) sekä uusi tekstitiedosto (*Tiedosto2*), johon listataan kaikki avainsanan osumat:

```
ack -l "Avainsana" -i --flush --files-from="Tiedosto1" > "Tiedosto2"
```

Tiedosto2 on siis väliaikaistiedosto, jota tarvitaan osumien poistamiseen *Tiedosto1*:stä. Näitä osumia on turha pitää enää *Tiedosto1*:ssä, koska nämä tiedostot tarkistettaisiin muuten uudestaan seuraavassa avainsanan haussa sekä skriptin muissa hauissa. Siksi

tämä komento tulee ajaa avainsanojen silmukan jokaisella iteraatiolla. Tähän tarvitaan vielä yhtä väliaikaistiedostoa (*Tiedosto3*), joka on iteraation avainsanoista vähennetty lista lähdekooditiedostoista. *Tiedosto1* voidaan lopuksi siis korvata *Tiedosto3*:lla:

```
grep -Fvxf "Tiedosto2" "Tiedosto1" > "Tiedosto3"
mv "Tiedosto3" "Tiedosto1"
```

4.3.4 Tiedostopolkujen sivuutukset

Eräs tapa tiedostopolkujen sivuutuksen helpottamiseksi hausta olisi poistaa tiedostoviitteet lähdekooditiedostojen listalta heti *Copyright*-haun jälkeen. Tämä ei kuitenkaan olisi tarkalleen ottaen ajateltuna tiedostopolkujen sivuutusta, vaan tiedostopolkujen siivousta jälkeenpäin. Jotta tiedostopolkuja voidaan sivuuttaa teknisesti oikein sekä suoritustehokkaasti, tämä tulee tehdä jo ack:n *Copyright*-haun aikana.

Ack:ssa on käytössä parametrivaihtoehto *--ignore-dir*, jota voidaan käyttää tiedostopolkujen sivuutukseen hakuvaiheessa. Ongelmana on, että tämä parametri pystyy käsittelemään vain yhden sivuutettavan tiedostopolun. Tämä on ongelmallinen tilanne, jos tietokantaan on tallennettu useampia sivuutettavia tiedostopolkuja. Tämä parametri voidaan kuitenkin syöttää yhteen ack-komentoon useamman kerran, joten tällä tavalla hakuun saadaan mukaan useampia sivuutuspolkuja:

```
ack -l -i --flush --cpp "Copyright" --ignore-dir=Tiedostopolku1 --ignore-
dir=Tiedostopolku2
```

Tämä toimintamalli toimisi, jos tiedostopolut olisivat aina samat eikä näitä tarvitsisi muuttaa. Tässä työssä tiedostopolut kuitenkin määritellään käyttöliittymän kautta ja tallennetaan tietokantaan. Nämä tiedostopolut täytyy siis hakea tietokannasta ja näiden lukumäärä saattaa vaihdella, joten tähän tarvitaan jälleen kerran *while*-silmukkaa:

```
ack -l -i --flush --cpp "Copyright" $( while IFS= read -r line;
do

      echo "--ignore-dir=$line";

done < <(echo 'SELECT name FROM ignore_folder' | mysql -uKäyttäjä
-pSalasana -dLisenssitietokanta | sed -n '1!p')) </dev/null > "Tie-
dosto1"
```

4.3.5 Avoimen lähdekoodin lisenssitekstin tulostaminen tiedostosta

Lisenssitekstin automaattista tulostamista varten täytyy ensin tunnistaa lähdekooditiedostosta oikea kommentointisyntaksi lisenssitekstistä. Tämä lisenssiehtojen ilmaisu löytyy siis usein kommentoituna lähdekooditiedoston alusta.

C++ -tiedostoissa kommentointi voidaan tehdä joko merkkiparilla (/* */);

```
/*
* Copyright © <Vuodet> <Tekijänoikeuden haltija>
* Permission is hereby granted, free of charge, to any person obtaining
* a copy of this software and associated documentation files (the
* "Software"),
  …
* ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
* OTHER DEALINGS IN THE SOFTWARE.
*/
```

tai yksittäisillä rivikommentoinneilla (//):

```
// Copyright © <Vuodet> <Tekijänoikeuden haltija>
//
// Permission is hereby granted, free of charge, to any person obtaining
// a copy of this software and associated documentation files (the
// "Software"),
  …
// ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
// OTHER DEALINGS IN THE SOFTWARE.
```

Merkkiparilla varustettu kommentointitapa ei vaadi ’*’-merkintää riveille, jotka ovat merkkiparien välissä, joten skriptissä on hyvä käydä yksittäisten rivikommenttien tarkastus ensin läpi. Jos *Copyright*–rivi sisältää kaksoisvinoviivan, voidaan kommentointisyntaksin olettaa noudattavan yksittäisten rivikommenttien tapaa:

```
if grep -i -n "Copyright" "Tiedostopolku" |  grep -F "//" | cut -f2- -d:
| awk '{print $1;}' | head -1 | grep -q -F -o "//"; then
COMMENT_SYNTAX="single"
```

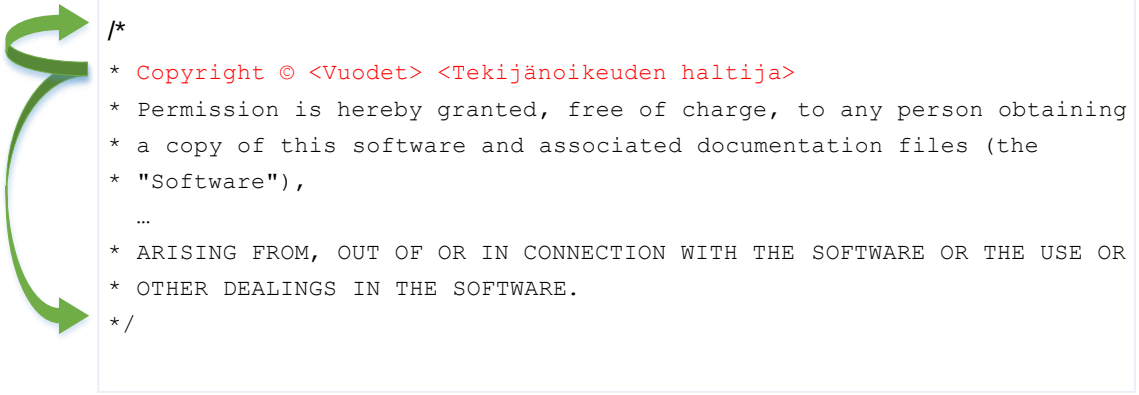
Linuxin *grep*–tulostus tulostaa konsoliin paljon ylimääräisiä merkkejä, joita ei tarvita tässä skriptissä. *Cut*–komennolla voidaan poistaa nämä turhat rivien alusta, jotta kommenttimerkinnän selvittäminen helpottuu skriptissä. Jollei rivin alusta löydy kaksoisvinoviivaa, niin skriptissä täytyy selvittää vielä merkkiparin mahdollisuus:

```
elif grep -i -C 40 "Copyright" "Tiedostopolku" | grep -q -F "/*"; then
COMMENT_SYNTAX="block"
```

Molempien kommenttityylien tulostus vaatii, että löydetään lähdekooditiedostosta lisenssitekstin ensimmäinen ja viimeinen rivi. Kun nämä rivit on paikannettu tiedosta, niin skriptissä voidaan tulostaa löydetty riviväli konsoliin.

Merkkipari rivivälin etsiminen

Merkkiparilla merkitty kommenttiväli on yksinkertainen löytää. Koska tällä tavalla komentoiduissa lisenssiteksteissä ei tarvitse kuin etsiä lähimmät merkkiparit *Copyright*–sanan ympäriltä. Kuvassa 10 esitetään graafisesti merkkiparin rivivälin hakeminen.



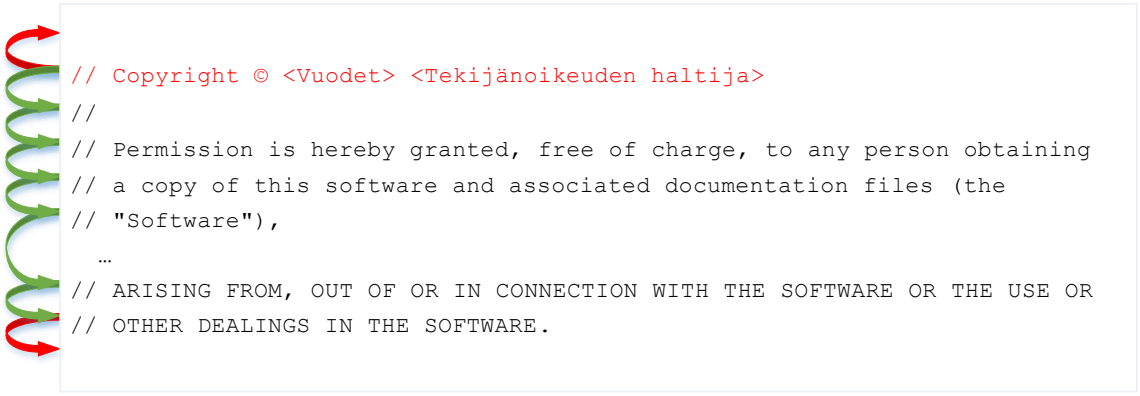
Kuva 10. Kommentointimerkkiparin hakemisen havainnollistus skriptissä.

Linuxin grep:ssä voidaan käyttää valitsinta –B, kun etsitään rivejä ennen *Copyright*-sanaa tai –A, kun etsitään rivin jälkeen olevia rivejä. Valitsimen jälkeen tulee laittaa myös rivimäärä, että montako riviä tarkastetaan korkeintaan:

```
(START tai END)=grep -i (-B tai -A) 40 -m 1 -n "Copyright" "Tiedostopolku"
| grep -i -n -F "/* jos -B ja */ jos -A" | cut -f1 -d- | cut -f1 -d: |
tail -1
```

Yksittäisten rivikommenttien rivivälien etsiminen

Yksittäiset rivikommentit käydään läpi etsimällä *Copyright*–sanan ympärillä olevien rivien alusta kaksoisvinoviiva. Tämä jälkeen tiedostosta täytyy yksitellen lähteä tarkastamaan lähirivejä rivikommenttien varalta. Rivejä käydään niin kauan läpi, ettei lähiriveiltä löydy enää rivikommenttimerkintää. Kuvassa 10 esitetään graafisesti yksittäisten rivikommenttien rivivälin etsiminen.



Kuva 11. Yksittäisten rivikommenttien hakemisen havainnollistaminen skriptissä

Toteutus tälle on hieman monimutkaisempi verrattuna merkkiparin ratkaisuun, koska tässä ei voida olettaa järkevästi lisenssitekstin päättymispaikkaa mitenkään. Merkkiparissa on turvallista olettaa, että *Copyright*-sanan lähimmät merkkiparit ylä- sekä alapuolelta pitävät sisällään vain tarvittavan lisenssitekstin. Tästä syystä skriptissä tarvitaan *while*-silmukkaa, jolla tarkistetaan lisenssitekstin päättymisehdot ylä- sekä alapuolelta:

```
$(START tai END)=grep -i -n "Copyright" "Tiedostopolku" | grep -i -n -F
"//" | cut -f1 -d- | cut -f1 -d: | head -1)
while read -r line
do
      if  echo $line | grep -i -q -F "//"; then
            $(START tai END)=$(echo $line | grep -i -F "//" | cut -f1 -
            d- | cut -f1 -d:)
      else
            break
      fi
done < <(grep -i $(-B tai -A) 40 -n "Copyright" "Tiedostopolku" | sed
'1!G;h;$!d')
```

Lopullinen tulostus

Kun skriptissä ollaan löydetty onnistuneesti lisenssitekstille alku- ja loppurivinumerot, niin näitä tietoja voidaan käyttää lisenssitekstin tulostamiseen. Linuxilla voidaan käyttää *sed*-komentoa, johon voidaan ajonyhteydessä syöttää suoraan tulostettavan rivivälin rivinumerot. Ja koska tämä skripti on käytössä vain käyttöliittymässä, joka käyttää HTML kieltä, niin tulostuksen yhteydessä täytyy myös lisätä rivien eteen <*br*>. HTML käyttää rivinvaihdossa eri merkkitunnistetta mitä C++-tiedostot, joten tulostuksen yhteydessä täytyy lisätä HTML:n oma rivivaihtotunniste eli <*br*>:

```
while read line;
    do
        echo "<br>$line"
done < <(sed -n "$START","$END"p "Tiedostopolku")
```

Tavallisesti rivivaihtomerkinnän vaihdon yhteydessä pitäisi myös poistaa toisen rivinvaihdontunniste, mutta linuxin echo poistaa tämän tulostuksen yhteydessä, niin tätä ei tarvitse tehdä tässä skriptissä.

4.3.6 Tallennus tietokantaan

Tunnistusskriptin jälkeen tiedot eivät ole tallennettu automaattisesti tietokantaan vaan tiedot on tallennettu väliaikaistiedostoihin. Syynä tälle on kehityksen mahdollistaminen tunnistus skriptille ilman vaarantamatta kaikkia tietokannassa olevaa dataa. Tiedostoihin on tallennettu eri listojen lähdekoodit riveittäin (verifoimattomat, verifioidut sekä musta lista). Nämä tiedostolistat voidaan siis käydä läpi silmukassa ja tallentaa tiedot tietokantaan. Ennen jokaista silmukkaa täytyy tyhjätä aikaisemmat listat sekä aloittaa taulujen automaattisesti inkrementoivien identiteettinumeroiden laskenta alusta:

```
echo "DELETE FROM lista tietokannassa" | mysql -uKäyttäjä
-pSalasana -dLisenssitietojanta
echo "ALTER TABLE lista tietokannassa AUTO_INCREMENT = 1" | mysql
-uKäyttäjä -pSalasana -dLisenssitietokanta
while read -r line
    do
      export fspec=$line
      fname=`basename $fspec`

      echo "INSERT INTO lista tietokannassa (filename, initial_path, re-
      pository_path) values('$fname', '$workspace', '$line')" | mysql
      -uKäyttäjä -pSalasana -dLisenssitietokanta

done < lista tiedostossa
```

4.3.7 Lisenssitiedoston generointi

Lisenssienhallinnan viimeinen komponentti on itse lisenssitiedoston generointi jatkuvan integraation prosessiketjun loppupäässä ennen ohjelmiston paketointi toimenpiteitä. Tämän tiedoston generointi vaatii kuitenkin vielä, että käyttäjä käy verifioimassa käyttöliittymän kautta verifioimattomaksi jääneet lähdekooditiedostot sekä ryhmittelemässä nämä lähdekooditiedostot saman lisenssin alla oleviin ryhmiin. Ryhmityksen jälkeen voidaan ajaa lisenssitiedoston generoiva skripti, joka hakee ryhmät tietokannasta ja tulostaa tiedostoon lisenssitekstin yhdestä ryhmän lähdekooditiedostosta viitteeksi:

```
while IFS= read -r line; do

      echo “Ryhmän nimi”

      SAMPLEFILE=$(echo "SELECT repository_path FROM license_file_data
      WHERE affliation='${line}'" | mysql -uKäyttäjä -pSalasana
      -dLisenssitietokanta | sed -n '1!p' | head -1)

      ./tulostusSkripti.sh $SAMPLEFILE | cut -f2 -d'>' | sed "s/^/||/"

done < <(echo “SELECT name FROM license_file_type” | mysql
-uKäyttäjä -pSalasana -dLisenssitietokanta | sed -n '1!p')
```

4.4 Käyttöliittymä avoimen lähdekoodin hallinnoinnille

Lisenssientunnistuksen tuloksena lisenssienhallinnassa on tietokannasta dataa, joka vaatii käsittelyä. Käyttöliittymässä suoritettavia toimenpiteitä ovat; listojen järjestely, lähdekooditiedostojen verifiointi sekä lähdekooditiedostojen ryhmittely. Lisäksi käyttöliittymässä täytyy pystyä hallitsemaan lisenssimalleja, avainsanoja sekä tiedostopolkujen sivuutuksia.

Verkkokäyttöliittymään liittyy neljä eri kieltä, joita käyttämällä saadaan tarvittavat toiminnot lisenssienhallintaa varten; HTML, CSS, Javascript sekä PHP. Näiden kielien avulla voidaan luoda tapahtumaketju, joka kuljettaa viestin käyttäjältä suoraan tietokantaan muutoksista joita halutaan tehdä esimerkiksi listoihin. HTML sekä CSS kielet ovat lähinnä vastuussa käyttöliittymän graafisesta ilmeestä ja tietokannan tulosten visuaalisesta esittämisestä. Tulosten esittämiseen tarvitaan kuitenkin Javascriptiä sekä PHP:ta, jotka toimivat viestiketjun kuljettavina kielinä. Tavallisesti HTML:n käyttäjäsyöte kenttien viestit kulkeutuvat ensin Javascriptille, jossa viestiä voidaan muokata PHP:lle käytettävään formaattiin. PHP:sta taas löytyy tarvittavat toiminnot, joilla viesti saadaan lopuksi tietokantaan tallennetuksi.

4.4.1 HTML-elementit

HTML-elementit ovat sivujen eri komponentteja, joita käytetään HTML-sivun muodostamiseen. Elementeillä on kullakin oma kokoelma attribuutteja, joilla voidaan muokata elementin tunnisteita sekä visuaalisia piirteitä. Vaikka elementeillä on useita kymmeniä attribuutteja niin näiden kaikkien arvoja ei tarvitse määrittää. Elementit sekä näiden eri attribuutit merkitään HTML-dokumentissa kulmasulkeiden sisään. Tässä dokumentissa käytetään värikoodausta elementtien sisällön hahmottamisen helpottamiseksi:

```
<elementti attribuuttil="attribuutin1 arvo" attribuutti2="attribuutin2 arvo">
```

Riippuen elementin käyttötarkoituksesta ja -kohdasta, elementti tulee myös katkaista käyttämällä komentoa *</elementti>*.

Link

Linkillä voidaan määrittää HTML -sivun oma CSS-tiedosto (.css), jota voidaan käyttää HTML sivun visuaalisten komponenttien muokkaamiseen. Tällä voidaan tehdä helposti usealle HTML sivulle oma standardi ulkonäkö kirjoittamalla ensin CSS-tiedoston standardi elementit ja käyttämällä *link* -elementtiä tämän jälkeen HTML -dokumentissa:

```
<link rel="stylesheet" type="text/css" href="tiedosto.css"></link>
```

Script

Jotta HTML sivu voi käyttää Javascript funktioita, niin käyttäjän täytyy linkata Javascript tiedosto (.js). Javascript tiedoston linkkaaminen toimii samalla periaatteella kuin CSS-tiedoston linkkaaminen, mutta elementin nimi on *Script*. Script ei vaadi muita attribuutteja kuin lähteen:

```
<script src="tiedosto.js" ></script>
```

IFrame

On kätevä HTML-elementti, jolla voidaan asettaa sivulle rajattu toinen dokumentti. HTML–sivuista on vaikea tehdä dynaamisia, koska useimpia elementtejä ei voi muuttaa sivun latauksen jälkeen ilman päivittämättä koko sivua. Tässä työssä käytetään tästä johtuen paljon Iframea, jolla saadaan helposti sivun tiettyjä elementtejä päivitettyä.

IFramella on monia atribuutteja, mutta tärkein näistä on src (source) eli lähde. Lähteellä voidaan päättää tiedosto, jota käytetään Iframessa. Tavallisesti tämä on toinen HTML-dokumentti, mutta iframen lähteenä voi käyttää myös PHP -tiedostoa:

```
<iframe src="tiedosto.(php/html)" id=”id_frame”>
```

Input

Input-elementillä saa HTML sivuun erilaisia syötekenttiä. Elementillä on useita erilaisia käyttäjäsyöte tyyppejä, joka määräytyy elementin *type*-attribuutin mukaan. Tyyppejä on

olemassa esimerkiksi; Painike, tekstikenttä sekä valintaruutu. Näiden nappuloiden toimintoja voi yhdistää javascript-tiedoston funktioihin:

```
<input type="button" onclick="Funktio(Parametri)" ></input>
```

4.4.2 Listojen esittäminen

Verkkokäyttöliittymän aloitussivu toimii käyttöliittymän ydinsivuna sekä listojen hallinnan työkaluna. Sivun tärkeimmät elementit ovat listojen esittämiseen tarkoitettu Iframe, kohde lähdekooditiedoston lisenssin esitys sekä navigointipalkki käyttöliittymän muille sivuille. Kuvassa 12 on esitettynä käyttöliittymän indeksisivu, josta voi muokata jatkuvan integraation lisenssientunnistuksen tuloksia.

Kuva 12. Käyttöliittymän aloitussivu

Listojen esittelyyn vaadittu elementti käyttää Iframea hyödykseen saadakseen dynaamisen näkymän lähdekooditiedostoille. Iframen lähteenä toimii PHP -tiedosto, joka hakee tietokannasta lähdekooditiedostot ja esittää nämä tiedot tulostettuina HTML-elementteinä. Tiedosto ottaa vastaan parametrina listan nimen, joka vastaa tietokannan pöydännimeä. Tämä parametrin syöttö voidaan taas puolestaan liittää suoraan sivun navigaatiopalkin painikkeisiin. Painikkeiden attribuutit voidaan kirjoittaa niin, että nämä vaihtavat Iframe-elementin lähteen, jotta tämä lähteeseen asetettu parametrin arvo vastaa painikkeen näyttämää listaa:

```
<a class="list_object" id="listan_id" name="Listannimi"
href="getlist.php?q=listannimi" target="nav_frame">Listannimi</a>
```

PHP-tiedostossa parametri voidaan ottaa vastaan käyttämällä $_GET-komentoa. Tämä parametri tallennetaan PHP-tiedostossa muuttujaan *$q,* ja tätä taas voi käyttää tietokannan kyselyssä. Kyselyn tulokset voidaan käydä läpi silmukalla. Silmukan jokaisella iteraatiolla täytyy listan lähdekooditiedostojen tulostamisen lisäksi tulostaa jokaiselle oma HTML-valintaruutu. Näille valintaruuduille täytyy asettaa omat uniikit arvot, jotta valintaruutujen painallukset voidaan myöhemmin tunnistaa siirron yhteydessä. Täksi arvoksi soveltuu siis mainiosti lähdekooditiedoston polku versiohallinnassa, joka on tallennettuna tietokantaan:

```
$sql = "SELECT filename, repository_path FROM $q";
$result = $conn->query($sql);

if($result->num_rows > 0)
{
    while($row = $result->fetch_assoc())
    {
        $repo_path = $row['repository_path'];
        $filename = $row['filename'];

        echo "<p><input type='checkbox' name='box' value=$repo_path><a
        id='noticeLink' href='#' name=$repo_path onclick='parent.update-
        Notice(this.name);return false;'>$filename</a></p>";
    }
}
```

Kirjoittamalla *onclick*-attribuutin lähdekooditiedostojen tulosteille Iframessa, tiedostojen nimet saadaan reagoimaan käyttäjän painalluksiin. Tätä voi siis käyttää tulostamineen lähdekooditiedostosta, jos Javascript -funktion ohjaa ajamaan Linuxin oikean skriptin. Javascript -funktio ei voi turvallisesti ajaa palvelimen shell-skriptejä, joten tätä varten on hyvä käyttää tälle tarkoitettua omaa PHP -tiedostoa (getnotice.php). Skriptin tulostama lisenssiteksti voidaan sijoittaa tälle varattuun paikkaan HTML -sivulta:

```
function updateNotice(filename)
{
    var xmlhttp = new XMLHttpRequest();
    xmlhttp.onreadystatechange = function()
    {
        if (this.readyState == 4 && this.status == 200)
        {
            document.getElementById("notice").innerHTML =
            this.responseText;
            document.getElementById("f_name").innerHTML = filename;
        }
    };
    xmlhttp.open("GET", "getnotice.php?q="+filename, true);
    xmlhttp.send();
}
```

Listojen tulostamisen lisäksi pitää kuitenkin vaihtaa lisäksi listojen siirtoon tarkoitettujen valintapainikkeiden nimet ja arvot. Listojen siirtäminen käyttää suoraan näiden valintaruutujen arvoja tietokantojen siirtojen määrittämiseen, joten nämä valintaruudut täytyy vaihtaa sellaisiksi, että Iframessa esitettävä lista ei ole näissä valintaruuduissa. Tähän käytetään yksinkertaista *switch-case*-funktiota, jonka parametrina käytetään *$q*-muuttujaa:

```
switch($q)
{
case "blacklist":
    $name_array=array("Verified", "Unverified", "Blacklist");
    break;
case "verified":
    $name_array=array("Blacklist", "Unverified", "Verified");
    break;
case "unverified":
    $name_array=array("Verified", "Blacklist", "Unverified");
    break;
default:
    exit();
}
```

Tämän tuloksena saadaan taulukko (*$name_array*), joka pitää sisällään arvot valitulle listalle sekä valintaruuduille. Taulukon arvot ovat asetettu vastaamaan pääsivun elementtejä niin, että arvot asettuvat sivulle oikein; ensimmäinen ja toinen arvo taulukossa vastaa

siirtovalintaruudukoiden arvoja ja kolmas arvo vastaa listan arvoa. Näitä arvoja voidaan tämän jälkeen käyttää Javascript-funktiossa, joka vaihtaa pääsivun elementtien arvot:

```
function updateInputForm(label1, label2, selected)
{
    document.getElementById("nav_frame").value = selected;
    document.getElementById("list1").innerHTML = label1;
    document.getElementById("list2").innerHTML = label2;

    var nodes = document.querySelectorAll('input[name="list"]');
    nodes[0].value = label1;
    nodes[1].value = label2;
}
```

4.4.3 Lähdekooditiedostojen siirtäminen

Listojen käsittely onnistuu elementeillä, jotka on kirjoitettu HTML-dokumentille sekä iframen esittämään listaan. Käyttäjän tulee ensin valita siirrettävät tiedostot listalta painamalla lähdekooditiedostojen edessä olevia valintaruutuja, jonka jälkeen tulee valita kohdelista ja painaa *Submit*-painiketta. Tämän operaation ketjun kaikki komponentit on jo kirjoitettu näiden esittämiseen vaadittaviin elementteihin, joten prosessiketjusta puuttuu vain tarvittavat javascript-funktiot sekä ajettavat PHP tiedostot.

Tietokantatasolla siirtäminen vaatii luettelon lähdekooditiedostoista. Luettelo voidaan muodostaa suoraan käyttäjän valitsemista tiedostoista iframen listan kautta. Javascriptillä voidaan etsiä kaikki iframen dokumentin *input*-elementit ja muodostamalla näiden elementtien arvoista taulukko:

```
var frame = document.getElementById("nav_frame");
var checkboxes = frame.contentWindow.document.getElementsByTagName("in-
put");

var elements[];

for (var i = 0; i < checkboxes.length; i++)
{
      if(checkboxes[i].checked)
      {
            var entryObj = {"fname" : checkboxes[i].value};
            elements.push(entryObj);
      }
}
```

Tämä taulukko täytyy lisäksi vielä muuntaa JSON:n syntaksia vastaavaksi merkkijonoksi komennolla *JSON.stringify(elements).* Tämä mahdollistaa luettelon vastaanottamisen PHP -tiedostossa, jossa merkkijono voidaan dekoodata PHP:n omiin muuttujiin.

Luettelon lisäksi PHP-tiedosto tarvitsee lähdelistan sekä kohdelistan. Lähdelistan saa suoraan iframen elementin arvosta, kun taas kohdelistaa varten täytyy tarkistaa listojen siirtovalinnoista, kumpi listoista on valittu siirron kohteeksi. Nämä arvot tulee myös muuttaa merkkijonoksi:

```
var fromValue = document.getElementById("nav_frame").value;
var toValue =
document.querySelector('input[name="list"]:checked').value;

var pathsObj = {"from" : fromValue, "to" : toValue};
var pathsParam = JSON.stringify(pathsObj);
```

Nämä merkkijonot voidaan nyt yhdistää ja ajaa PHP–skriptissä (*move_entries.php*). Tämän täytyy ensin dekoodata Javascriptin kautta vastaanotettu JSON –merkkijono ja siirtää vastaanotetut lähdekooditiedosto viitteet omaan taulukkoon. Tämä onnistuu yhdellä PHP–komennolla:

```
$taulukko=json_decode($jsonMerkkijono, true);
```

Tämä taulukkoa voidaan nyt käydä läpi silmukassa. Silmukassa täytyy kerätä lähdelistan lähdekooditiedostojen arvot muuttujiin, ja lisätä nämä tietokanta–kyselyillä kohdelistaan. Tämän lisäksi nämä tiedostot täytyy aina onnistuneen kyselyn jälkeen poistaa vanhasta listasta, jotta listoihin ei muodostu duplikaatteja:

```
foreach($paramsObj as $obj)
{
    $filestr = $obj['fname'];
    $sql = "SELECT * FROM $lähdelista WHERE repository_path='$filestr'";
    $result = $conn->query($sql);

    if($result->num_rows > 0)
    {
        $row = $result->fetch_assoc();
        $repo_path = $row[ 'repository_path'];
        $filename = $row['filename'];
        $initial_path = $row['initial_path'];

       $sql = "INSERT INTO $kohdelista (filename, initial_path, repository_path)
       VALUES ('$filename', '$initial_path', '$repo_path')";

        if($conn->query($sql) === TRUE)
        {
            $sql = "DELETE FROM $lähdelista WHERE repository_path='$filestr'";
            if($conn->query($sql) === FALSE)
            {
                    exit;
            }
        }

        else
        {
            exit;
        }
    }
}
```

4.4.4 Hakukriteerien esittäminen

Hakukriteerien hallinnalle on oma HTML-sivu, josta voi helposti hallita kaikki hakukriteereitä sekä rajoitteita. Hallittavia kohteita ovat lisenssimallit, avainsanat sekä tiedostopolkujen sivuutukset. Näitä kaikkia täytyy pystyä lisäämään, muokkaamaan sekä poistamaan käyttöliittymän kautta. Jokaisella näillä hakukriteereistä on kuitenkin erilaiset pöytärakenteet tietokannassa, joten jokaiselle näistä täytyy tehdä omat PHP -tiedostot sekä Javascript-funktiot. Kuvassa 13 on esitettynä hakukriteerien hallintasivu.

Kuva 13. Hakukriteerien hallintasivu

Sivun painikkeet

Sivu käyttää painikkeita, jotka eivät ole HTML:n omia *input*-painikkeita. Näiden painikkeiden avulla käyttäjä voi valita lisenssimallien -ja avainsanojen listatyypit. Valitsemalla listatyypin, käyttäjä muuttaa sivun tiputuslaatikoiden tulokset tietokannan listojen mukaiseksi. Painikkeiden graafinen ulkonäkö saadaan kirjoittamalla näille oma luokan kuvaus CSS-tiedostoon:

```
.type_object
{
    display: block;
    background-color: #00008B;
    color: white;
    text-align: center;
    padding: 12px 14px;
    text-decoration: none;
}
```

Kirjoittamalla tämän kuvauksen, HTML -dokumentissa voi nyt käyttää *type_object*–nimistä luokkaa elementtinä, joka perii CSS–tiedostosta tälle luodun graafisen kuvauksen. Tämän lisäksi painikkeille on hyvä luoda myös oma listaus profiili. *<ul>* -elementillä (unordered list) painikkeet saa helposti järjestykseen ilman kirjoittamatta jokaiselle painikkeelle sijainteja manuaalisesti HTML–dokumenttiin:

```
ul.types
{
   list-style-type: none;
   margin: 0;
   padding: 0;
   width: 100%;
   overflow: hidden;
}
```

Listojen vaihtaminen

Sivulla on yhteensä viisi eri listaa muokattavissa. Kaksi lisenssienmallien listaa, kaksi avainsanojen listaa sekä tiedostopolkujen sivuutukset. Sivulla voi kuitenkin olla näkyvillä jokaisesta kategoriasta yksi lista kerrallaan. Nämä hakukriteerit ovat näkyvillä tiputuslaatikoissa, jotka esitetään pienen *iframe*-elementin kautta. Listojen vaihtaminen tapahtuu siis painamalla tiputuslaatikoiden yläpuolella olevia painikkeita, joihin on kirjoitettu Javascript–funktio *onclick*–attribuutin avulla. Tämän funktion tehtävä ei ole muuta kuin välittää iframen PHP–dokumentille parametrina tietokannan taulukon nimi, joka ladataan tiputuslaatikkoon. Tämä täytyy tehdä jokaiselle hakukriteeri kategorialle erikseen taulukoiden erilaisuuksista johtuen:

```
function getKategoriaElements(type, buttonid)
{
    var frame = document.getElementById("dropdown_frame");
    frame.src = "getKategoria.php?table="+type;
}
```

Tiputuslaatikoiden sisältöjä vaihtavien PHP–skriptien täytyy ensinnäkin hakea tietokannasta valmiit hakukriteerit ja esittää nämä taulukon jonojen nimet tiputuslaatikossa. Avainsanoille sekä tiedostopolun sivuutuksille riittää esittämiseen pelkkä tiputuslaatikon näkymä, koska näiden taulukoissa ei ole muita käyttäjälle olennaisia kenttiä kuin nimi:

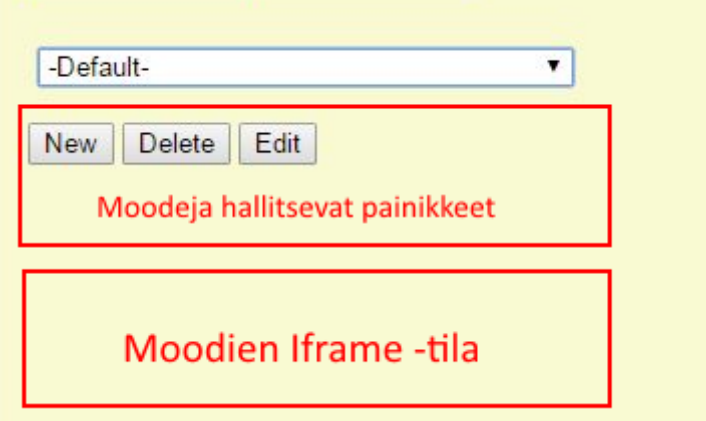
```
$sql = "SELECT * FROM $pöydännimi";
$result = $conn->query($sql);

echo "<br><select style='width: 20em;'name=$pöydännimi id='hakukatego-
ria_dropdown'>";
echo "<option id=default value='default'>-Default-</option>";
if($result->num_rows > 0)
{
    while($row = $result->fetch_assoc())
    {
        $name = $row['name'];
        $id = $row['id'];
        echo "<option id=$id value=$name>$name</option>";
    }
}

echo "</select>";
```

4.4.5 Hakukategorioiden moodit

Jokaiselle kategorialle on varattuna oma *iframe* moodeja hallitsevien painikkeiden alapuolelle. Poikkeuksena lisenssimallit, jolle on varattuna oma tila sivun keskiosaan. Iframeen tuodaan näkyviin jokaiselle moodille tarkoitetut elementit, jos käyttäjä haluaa muokata, poistaa tai luoda uusia arvoja tietokantoihin. Iframen lähteenä käytetään custom_*kategoria*_frame.php tiedostoja, joiden parametreillä voidaan välittää tarvittavia tietoja. Kuvassa 14 on esitelty moodien hallintapainikkeet ja niiden vaikututtama alue sivulla.



Kuva 13. Kategorioidenhallinnan tilat esitettynä

Tärkein parametri näistä on *mode*, jolla voidaan vaihtaa moodien hallinnan näkymää tulostamalla eri elementtejä. Moodeja hallitseviin painikkeisiin on liitetty Javascript–funktio, joka saa eri parametrin riippuen painikkeesta (*edit*, *delete*, *edit*):

```
<input type="button" value="New" onclick="switchKategoriaMode('new')">
<input type="button" value="Delete" onclick="switchKategoriaMode('delete')">
<input type="button" value="Edit" onclick="switchKategoriaMode('edit')">
```

Javascript–funktio saa vain vaihdettavan moodin parametrinä, joten funktiossa täytyy tämän lisäksi hakea myös tiputuslaatikon valitun objektin identiteettinumero sekä tietokannan pöydänimi. Tämän jälkeen nämä arvot välitetään custom_*kategoria*_frame.php–skriptille parametreinä:

```
function switchKategoriaMode(mode)
{
	var frame = document.getElementById("kategoria_frame");
	var dropdown = document.getElementById("custom_kategoria_frame").conten-
	tWindow.document.getElementById("kategoria_dropdown");
	var tablename = dropdown.name;
	var id = dropdown.options[dropdown.selectedIndex].id;
	var source = "custom_template_frame.php?mode=" +mode+ "&id=" +id+ "&ta-
	ble=" +tablename;
	frame.src = source;
}
```

PHP –skriptissä voidaan käyttää *switch-case*–funktiota käyttämällä *modea* parametrina. Tällä tavalla iframeen saadaa tulostettua moodia vastaavat elementit ja täyttämällä elementtien arvot oikein. Elementtejä tulostaessa täytyy ottaa huomioon moodi sekä kategoria:

```
switch($object['mode'])
{
case "new":
    $id = 0;
    echo "
       <input type='text' style='width: 30em' id=0 value=''></input><br>
       <input type='button' value='Save' onclick='saveKategoria($id)'></input>
       <input  type='button'  value='Cancel'  onclick='resetCustomFrame()'></in-
       put>";
    break;

case "delete":
    echo "
       Are you sure you want to delete $objekti?
       <input type='button' value='Yes' onclick='deleteKategoria($id)'></input>
       <input  type='button'  value='Cancel'  onclick='resetCustomFrame()'></in-
       put>";
    break;

case "edit":
    echo "
       <input  type='text'  style='width:  30em'  id=$id  value='$license'></in-
       put><br>
       <input type='button' value='Save' onclick='saveKategoria($id)'></input>
       <input  type='button'  value='Cancel'  onclick='resetCustomFrame()'></in-
       put>";
    break;

default:
    exit;
}

echo "<script src='kategoria_customization.js'></script>";
```

Avainsanojen sekä tiedostopolkujen sivuutuksessa riittää pelkkä tekstikenttä muokkausta sekä uutta luotaessa. Lisenssienmallit tarvitsevat tämän lisäksi vielä lisenssitekstille oman kentän ja lisämoodin (*display*) lisenssitekstin esittämistä varten. Lisenssiteksti voidaan esittämisen yhteydessä näyttää pelkällä <*p*>-elementillä, mutta muokkaamista ja uusien lisäämistä varten <*input type='text'*>-syöttökenttä ei riitä. Tämän elementin näkymä ei ole suunniteltu rivinvaihdoille, joita lisenssiteksteissä esiintyy normaalisti. Pitkiä tekstejä varten tulee käyttää <*textarea*>-elementtiä.

Jokaiselle kategorialle on kirjoitettu myös omat Javascript–tiedostot, joiden funktioiden komennot vastaavat kategorian tietoja sekä PHP–skriptejä.

Poistamisprosessi

Objektien poistaminen on yksinkertainen prosessi. Tämä prosessi ei vaadi muuta kuin poistamiseen tarkoitetut Javascript–funktion sekä PHP–skriptin. Tietokannasta poistamiseen ei tarvita muuta kuin pöydännimi sekä identiteettinumero. Identiteettinumero voidaan välittää funktiolle parametrina ja pöydännimi voidaan hakea tiputuslaatikon arvoista. Nämä tiedot ajetaan AJAX:lla, joka suorittaa PHP–skriptin ja lataa tiputuslaatikon iframe uudestaan skriptin suorituksen jälkeen:

```
var xmlhttp = new XMLHttpRequest();
xmlhttp.onreadystatechange = function()
{
       if(this.readyState == 4 && this.status == 200)
       {
              parent.document.getElementById("kategoria_frame").contentWindow.
              location.reload(true);
        }
};

var table = parent.document.getElementById("kategoria_frame").contentWindow.doc-
ument.getElementById("kategoria_dropdown").name;
var parameters = "table="+table+"&id="+id;
xmlhttp.open("POST", "delete_kategoria.php", true);
xmlhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
xmlhttp.send(parameters);
```

PHP–skriptissä ei tarvitse muuta kuin suorittaa tietokanta kysely ja poistaa identiteettinumeroa vastaava rivi tietokannan taulukosta:

```
$sql = "DELETE FROM $table WHERE id=$id";

if($result = $conn->query($sql))
{
    echo "$name deleted from database successfully";
}
```

Tallennusprosessi

Kun moodien hallinnasta valitaan editoinnille tai uusien luomiselle tarkoitetut elementti –sarjat, niin nämä elementit sisältävät objektin tallennukseen johtavat funktiot. Muokkaus ja uuden luominen käyttävät molemmat samaa *save**Kategoria**(id)*–funktiota, joka ajaa tietokantaan tallentavan PHP-skriptin. Tämä funktio on liitettynä *Save* –painikkeeseen kun moodiksi on valittu uuden luonti tai muokkaus. Funktion kautta välitetään PHP–skriptille tallennettavan objektin pöydännimi, identiteettinumero, objektin nimi sekä lisenssimallin tapauksessa myös lisenssiteksti. Nämä tiedot voidaan välittää PHP–skriptille samalla tavalla AJAX:lla kuin poistamisen yhteydessä.

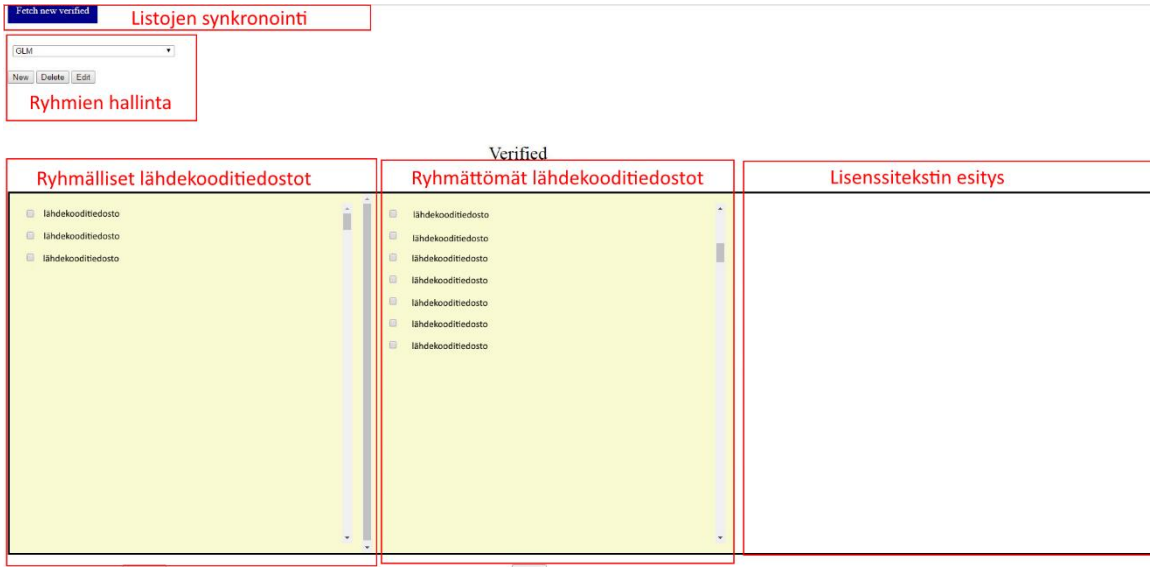
Uusia objekteja luotaessa identiteettinumerona käytetään numeroa 0. Tietokannassa identiteettinumeroiden laskenta aloitetaan numerosta 1, joten 0 -numeroa voidaan käyttää PHP–skriptissä tunnisteena uudelle riville tietokantaan:

```
if($id == 0)
{
	$name = $conn->real_escape_string($name);
	$sql = "INSERT INTO $table (name) VALUES ('$name')";
}
else
{
	$name = $conn->real_escape_string($name);
	$sql = "UPDATE $table SET name='$name' WHERE id=$id";
}

$conn->query($sql)
```

4.4.6 Lähdekooditiedostojen ryhmienhallinta sivu

Käyttöliittymän etusivun kautta tehtävän lähdekooditiedostojen verifioinnin jälkeen, verifioidut lähdekooditiedostot voidaan ryhmitellä tälle tarkoitetulla sivulla. Sivun kautta käyttäjän pystyy luomaan uusia ryhmiä ja siirtämään lähdekooditiedostoja näihin ryhmiin tai ryhmästä pois. Ryhmättömien ja ryhmällisten esittämistä varten sivulla on kaksi listaa, joiden kautta voi seurata helposti listojen lähdekooditiedostoja rinnakkain. Listojen tiedostonimet ovat interaktiivisia, joten lisenssit saa näkyviin klikkaamalla näitä. Listat ovat esitettynä Iframejen kautta. Oikeanpuoleinen iframe esittää aina verifioitujen lähdekooditiedostojen listan ja vasemmanpuoleinen lista esittää valitun ryhmän listan. Ryhmien listoja kontrolloidaan listan yläpuolella olevasta tiputusvalikosta, joiden objektienhallinta onnistuu samalla tavalla kuin hakukriteerienhallinta sivulla olevien lisääminen, muokkaaminen ja poistaminen. Kuvassa 15 on esitettynä ryhmienhallinta sivu, jossa lisensioidut lähdekooditiedostot sidotaan saman lisenssiryhmän alle.



Kuva 15. Ryhmienhallinnan sivun elementit esiteltynä.

4.4.7 Lisenssiryhmien synkronointi

Jotta verifioidut tiedostot näkyvät tälle varatussa tilassa, niin tämä lista täytyy ensin synkronoida. Sivulla esitetty verifioitujen lista ei esitä suoraan *verified*-taulukkoa tietokannasta vaan ryhmittelyyn tarkoitetun taulukon (*license_file_data)*, jonka tietokannan rivit täytyy synkronoida verifioitujen omasta taulukosta. Synkronointi voidaan suorittaa *Fetch new verified*-painikkeesta sivulta, joka ajaa Javascript funktion *fetchNewVerified()* sivun omasta Javascript-tiedostosta:

```
var xmlhttpx = new XMLHttpRequest();
xmlhttpx.onreadystatechange = function()
{
	document.getElementById("frame").contentDocument.location.re-
	load(true);
}

xmlhttpx.open("GET", "fetchverified.php", true);
xmlhttpx.send();
```

PHP-skriptissä synkronointi suoritetaan lisäämällä verifioitujen listalla olevia rivejä yksitellen lisenssiryhmien taulukkoon. Jokaisen rivin olemassaolo täytyy kuitenkin tarkistaa lisenssiryhmien taulukosta, jotta valmiiksi taulukossa olevia rivejä ei duplikoida taulukkoon:

```
$initial_path = $row[ 'initial_path'];
$repo_path = $row[ 'repository_path'];
$filename = $row[ 'filename'];

$sql =
"INSERT INTO license_file_data (filename, initial_path, repository_path)
SELECT * FROM (select '$filename', '$initial_path', '$repo_path') as tmp
where not exists(  select repository_path from license_file_data where
repository_path = '$repo_path' ) limit 1;";

$conn->query($sql)
```

Lisäämisen lisäksi täytyy huolehtia myös vanhojen rivien poistamisesta. Joskus saattaa olla, että joitain lähdekooditiedostoja on verifioitu ja synkronoitu lisenssiryhmien pöytään, mutta nämä tiedostot on myöhemmin poistettu tai siirretty verifioitujen pöydästä:

```
$sql = "DELETE FROM license_file_data WHERE repository_path NOT IN (SE-
LECT f.repository_path from verified f);"

$conn->query($sql)
```

4.4.8 Ryhmäobjektienhallinta

Ryhmien lisääminen, muokkaus sekä poistaminen noudattavat samaa kaavaa kuin hakukriteerien hallinta. Lisäämiseen tarvittavat elementit, funktiot sekä skriptit ovat kirjoitettu lähes samalla tavalla. Tietokannassa ryhmille ei ole kirjoitettu omaa kenttää, joka pitäisi kirjaa ryhmään lisätyistä lähdekooditiedostoista. Lähdekooditiedostojen ryhmittely tapahtuu vain lisenssiryhmien listataulukossa (*license_file_data)*, joten tämä täytyy ottaa huomioon, kun ryhmiä muokkaa tai poistaa:

Muokkaaminen

Muokkaamisessa täytyy tehdä samaan aikaan kaksi kyselyä; päivittäminen lisenssiryhmiin sekä lisenssiryhmien listaan. Lisenssiryhmien listasta vaihdetaan lähdekooditiedostojen *affliation*-kentän sisällöt, joiden arvo vastaa lisenssiryhmän vanhaa nimeä:

```
$sql = "SELECT * FROM license_file_type WHERE id='$id'";
$result = $conn->query($sql);

if($result->num_rows > 0)
{
	$row = $result->fetch_assoc();
	$oldname = $row['name'];

	$sql = "UPDATE license_file_type SET name='$name' WHERE id=$id";
	$sql2= "UPDATE license_file_data SET affliation='$name' WHERE af-
	fliation='$oldname'";

	if(!$conn->query($sql) || !$conn->query($sql2))
	{
		echo "Error while updating database";
	}
}
```

Poistaminen

Kun lisenssiryhmä poistetaan tietokannasta, niin tähän ryhmään kuuluvien lähdekooditiedostojen *affliation*-arvo täytyy nollata lisenssiryhmien listasta. PHP–skriptissä on tärkeä ottaa lisenssiryhmän nimi ylös ennen poistamista, jottei viite tästä ryhmästä katoa:

```
$result = $conn->query( "SELECT name FROM license_file_type WHERE
id=$id");

$name = $result->fetch_assoc()[ 'name'];

$sql = "DELETE FROM license_file_type WHERE id=$id";

if($result = $conn->query($sql))
{
	$sql = "UPDATE license_file_data SET affliation=NULL WHERE afflia-
	tion='$name'";

	$conn->query($sql))
}
```

4.4.9 Lähdekooditiedostojen ryhmittely

Kun käyttäjä on luonut ryhmäobjektin ryhmienhallintaan varatusta tilasta, niin tämä ilmaantuu tallennuksen jälkeen sivun pudotusvalikkoon. Valitsemalla pudotusvalikosta ryhmän, tämä vaihtaa sivulla olevan iframen esittämään ryhmässä olevat lähdekooditiedostot käyttämällä *onchange*–attribuuttia pudotuslaatikon elementissä. Tämä *switchList()* Javascript–funktio antaa listan iframelle parametrina ryhmännimen, joka puolestaan ajaa

PHP–skriptin ja hakee ryhmän kaikki lähdekooditiedostot ryhmien listasta. Hakemisen jälkeen nämä tulostetaan tuttuun tapaan valintalaatikon sekä lähdekooditiedoston nimen kanssa iframeen:

```
$q = $_GET["q"];

if($q == "")
{
	$sql = "SELECT filename, repository_path FROM license_file_data WHERE af-
	fliation IS NULL";
}

else
{
	$sql = "SELECT filename, repository_path FROM license_file_data WHERE af-
	fliation='$q'";
}

$result = $conn->query($sql);

if($result->num_rows > 0)
{
    while($row = $result->fetch_assoc())
    {
        $repo_path = $row['repository_path'];
        $filename = $row['filename'];

         echo "<p><input type='checkbox' name='box' value=$repo_path><a id='no-
         ticeLink'     href='#'     name=$repo_path     onclick='parent.update-
         Notice(this.name);return false;'>$filename</a></p>";
    }
}
```

Juuri luodulla ryhmällä ei tietenkään ole aluksi yhtään lähdekooditiedostoa listassa, joten tähän ryhmään halutut lähdekooditiedostot täytyy siirtää tämän viereisestä listasta painamalla haluttujen lähdekooditiedostojen kohdalla olevaa valintaruutua ja painamalla sivun alareunasta *Submit*–painiketta. Submit painike ajaa *submitFiles()*–funktion, joka hakee valintaruutujen arvot ja antaa nämä parametrina *submit_files.php*–skriptille. Samaa toiminta periaatetta käytetään lähdekooditiedostojen listojen siirtämisessä, mutta lähdekooditiedostojen pöytien siirtämisen sijaan muokataan ryhmittelylistan *affliation*–arvoja lisenssiryhmien listan taulukosta:

```
foreach($objektientaulukko as $obj)
{
	$filestr = $obj['fname'];

	$sql = "UPDATE license_file_data SET affliation='$kohderyhmä' WHERE
	repository_path='$filestr'";

	$result = $conn->query($sql);
}
```

Tiedostojen poissiirtäminen onnistuu valitsemalla ryhmän listasta olevat lähdekooditiedostot ja painamalla *Unsubmit*–painiketta. Joka käyttää taas *unsubmitFiles()*–funktiota ja

unsubmit_files.php–skriptiä. Toimintoprosessi on sama, mutta tiedostojen *affliation*–arvot taulukosta nollataan:

```
$sql = "UPDATE license_file_data SET affliation=NULL WHERE repository_path='$filestr'";

$conn->query($sql)
```

5 Jatkokehitys

Työllä on valtavasti jatkokehityspotentiaalia. CI-ympäristön kehitys kulkee rinnakkain itse tuotteen kehityksen kanssa, joten CI-ympäristön kehitys voi kuulua yrityselämässä joko yhdelle henkilölle tai tämä voi olla kehittäjien yhteinen tavoite. Tämän kyseisen työn jatkokehittämisessä parasta on, että kehityskohteena ei tarvitse olla koko työ vaan työstä voi kehittää yksittäisiä komponentteja ajan myötä, kuten lisenssienhallinnan käyttöliittymä.

Työn ohella ilmaantui monenlaisia ideoita, joita työstä voisi mahdollisesti kehittää tulevaisuudessa. Osa näistä kehitysideoista ei välttämättä ole edes kovin laajoja ja aikaa vieviä muutoksia, mutta työ laajuudesta johtuen työstämisen pääpaino oli saada toimiva rakenne CI-ympäristölle ja integroidulle lisenssienhallinnalle.

5.1 CI-ympäristön jatkokehitys

5.1.1 Yhteinen tallennustila käännetyille ohjelmistoille

Testaaminen on tärkeä osa ohjelmiston kehitysprosessia, ja tätä tapahtuu jatkuvasti monella eri tapaa. CI-ympäristö hoitaa paljon automatisoituja kriittisiä testejä, mutta välillä testaajana haluaa toimia esimerkiksi kehittäjä tai asiakas. Tätä varten käännetyt ja testatut ohjelmistot on hyvä olla jossain tallennustilassa, josta nämä on helppo saada. Jenkins tekee omista käännöksistään artefakteja, jotka tallennetaan Jenkinsin omiin tiedostopolkuihin. Näiden artefaktien ja niiden kanssa käytettyjen testiresurssien siirtäminen universaalimpaan tallennustilaan helpottaisi manuaalisen testiympäristö käyttöönottamista yrityksen sisäisesti.

5.1.2 Pipeline pikaista käyttöönottoa varten

Jenkinsin ei tarvitse olla pelkästään automatisoitu kääntö- ja testausympäristö. Käytännössä Jenkinsiin voi luoda myös käyttöönottoa varten oman pipelinen, joka hyödyntäisi käännettyjä ohjelmistokoosteita pikaista käyttöönottoa varten. Ohjelmistot saattavat joskus olla erittäin laajoja ja monimutkaisia, joten pelkkä ohjelmiston purkaminen ei riitä testiympäristön ohjelmiston toimintaan. Tämän prosessin voi kuitenkin tehdä yksinkertaiseksi

käyttäjälle hyvällä pipelinella, johon käyttäjän ei tarvitse muuta kuin valita ohjelmistonversio, ohjelmistonresurssit sekä kohdeympäristö pudotuslaatikoista. Tämä onnistuu Jenkinsin parametreillä.

5.2 Lisenssien tunnistuksen jatkokehitys

5.2.1 Lisenssientekstien tunnistus algoritmin parantaminen

Lisenssientunnistuksen skriptin kehityksen aikana ensimmäinen ajatus lisenssitekstien tunnistukseen liittyen, oli tarkastaa lisenssimalleja käyttäen rivi kerrallaan. Tämä osoittautui lopuksi huonoksi ideaksi, koska lähdekooditiedostoissa rivinvaihto saatetaan suorittaa eri kohdassa mitä lisenssimallissa saattaa olla. Tätä ongelmaa varten olisi kehitettävä jokin vertaus algoritmi, joka ei huomioi rivinvaihtomerkkiä lähdekooditiedostosta. Tällä tavalla lisenssimallin rivien merkkijonoja voisi verrata suoraan lähdekooditiedostoihin.

5.2.2 Lisenssien tunnistus osaksi kääntämisprosessia

Tämänhetkinen systeemi luo lisenssitiedoston käyttäjän luomien ryhmien perusteella. Nämä ryhmät saattavat olla kirjastoja, jotka eivät välttämättä ole edes ohjelmiston julkaisuversiossa mukana, joten tiedot näistä päätyvät lisenssitekstitiedostoon. Ideaalissa tilanteessa lisenssientunnistus komponentti voitaisiin suorittaa kääntäjän kanssa yhdessä. Tähän tarvittaisiin kääntäjältä jokaisen käännetyn ohjelman tai kirjaston jälkeen lista kaikista käännetyistä lähdekooditiedostoista, jotka tarkistettaisiin jollain skriptillä avoimenlähdekooditiedostojen varalta. Jos lisenssienhallinnasta haluttaisiin rakentaa täysin automatisoitu, niin tämä toimintaperiaate olisi mahdollinen lähtökohta tällaiselle systeemille.

5.3 Lisenssienhallinnan käyttöliittymä

5.3.1 Graafiset päivitykset

Käyttöliittymän visuaalisen ilmeen valinta ei perustu minkäänlaisiin psykologisiin tutkimuksiin värien vaikuttamisesta ihmismielentoimintaan ja erilaisten aatteiden edustamiseen. Käyttöliittymä on osana paljon laajempaa kokonaisuutta tässä insinöörityössä, joten käyttöliittymä ei ylpeile erityisesti omalla kauneudellaan. Tässä työssä pääpaino on ollut käyttöliittymän toiminnallisuudessa ja viestien käsittelyssä tietokannan kanssa. Käytännössä käyttöliittymän voi kirjoittaa ihan uudelleen käyttäen joko verkkopalvelinratkaisua tai omaa ohjelmaansa.

5.3.2 Tiputuslaatikot pois Iframeista

Tällä hetkellä käyttöliittymän eri skriptit ja viittaukset iframeihin ovat osittain hyvin vaikealukuisia. Varsinkin tiputuslaatikoiden sijoittaminen iframeihin saattaa tuottaa vaikealukuisuutta, koska Javascripteissä täytyy hakea oikeat viitteet iframeihin ja näiden sisältämiin dokumentteihin. Ensimmäinen ajatus tässä ratkaisumallissa oli saada tiputuslaatikoiden sisällöt muuttumaan ilman päivittämättä itsesivun päädokumenttia. Tämä onnistuu kuitenkin ilman iframea sekä sivun päivittämistä AJAX:n avulla.

6 Yhteenveto

Kokonaisuudessaan työ oli onnistunut pääominaisuuksien puolesta. Työn tuloksena saatiin aikaiseksi CI-ympäristölle päivitetty pipeline, joka hyödynsi päivityksen uusimpia ominaisuuksia ja vähensi iteraatioiden kestoa huomattavasti. Lisäksi päivitys toi paljon helppokäyttöisemmän näkymän ongelmista, joita iteraatioissa tulee vastaan. Tämä puolestaan taas nopeuttaa paljon lähdekooditiedostojen korjausta.

Työsuunnitelmaan kuului, että työssä pyrittäisiin parantamaan myös testiautomaatioympäristöä. Tämä kuitenkin jäi työssä hieman taka-alalle, koska tämä olisi monimutkaistanut vielä työtä paljon enemmän tuomalla uusia ohjelmointikieliä lisää. Testiautomaatio on osa CI-ympäristöä, mutta pohjimmiltaan tämä on oma aihealueensa työssä. Testiautomaatioympäristöä pystyisi mahdollisesti lähteä kehittämään ilman tutkimatta CI-ympäristön toiminta periaatteita tai lisenssienhallintaa.

Työ tarjosi valtavan määrän uutta opittavaa sekä haasteita uusissa ympäristöissä. Oppimiskokemuksena opinnäytetyö oli loistava, koska työ ei niinkään keskittynyt pieniin yksityiskohtiin vaan työllä oli laaja skaala, joka auttoi hahmottamaan suuria kokonaisuuksia. Tämä auttoi puolestaan ymmärtämään paremmin aikaisemmassa vaiheessa koulua opittua asiaa, jota ei osannut hahmottaa kurssien aikana. Lisäksi työ paransi valtavasti valmiuksia työelämään tarjoamalla työympäristön, joka auttoi sisäistämään intranetin kanssa toimimista yrityselämässä.

Lähteet

(1) Joonas Koski. Ketterät menetelmät, agile, LEAN ja scrum. Viitattu 25.1.2017. https://www.itewiki.fi/opas/ketterat-menetelmat-agile-lean-ja-scrum/

(2) Martin Fowler, 2006. Continuous Integration. Viitattu 25.1.2017. https://www.martinfowler.com/articles/continuousIntegration.html

(3) Juuli Kiiskinen, 14.3.2017. Moderni ohjelmistokehitys – vesiputousmalli vs. ketterät menetelmät. Viitattu 26.1.2018. http://blogi.sysart.fi/moderni-ohjelmistokehitys-pahkinankuoressa-vesiputousmalli-vs.-ketterat-menetelmat

(4) Margaret Rouse, Heinäkuu 2017. Agile Software Development. Viitattu 31.1.2018. http://searchsoftwarequality.techtarget.com/definition/agile-software-development

(5) Scrum Alliance. What is scrum?. Viitattu 2.2.2018. https://www.scrumalliance.org/why-scrum

(6) Sininen Meteoriitti, 2013. Ketteryys Haltuun: Yleisimmät ketterät käytännöt. Viitattu 4.2.2018. https://www.meteoriitti.com/2013/06/06/ketteryys-haltuun-yleisimmat-ketterat-kaytannot/

(7) Jez Humble & David Farley, 2010. Continuous Delivery: Anatomy of the Deployment Pipeline. Viitattu 27.1.2017. http://www.informit.com/articles/article.aspx?p=1621865

(8) Juhana Huotarinen, 20.4.2016. Tiesitkö jutkuvan julkaisun olevan jo arkipäivää. Viitattu 11.12.2017. https://gofore.com/tiesitko-jatkuvan-julkaisun-olevan-jo-arkipaivaa/

(9) Yegor Bugayenko, 2014. Continuous Integration is Dead. Viitattu 26.1.2017. http://www.yegor256.com/2014/10/08/continuous-integration-is-dead.html

(10) Jussi Ahonen, 2012. Laatu ja testaus: Automatisointi. Viitattu 26.1.2017 http://testausosy.fi/wp-content/uploads/2012/11/LT-Vol1Ed2.pdf

(11) Kyle McMeekin, 1.11.2017. Test Automation vs. Automated Testing: The Difference Matters. Viitattu 16.2.2018. https://www.qasymphony.com/blog/test-automation-automated-testing

(12) Jussi Pekka Kasurinen. Ohjelmistotestauksen käsikirja. Docendo Oy, 2013. Viitattu 23.2.2018

(13) Matti Korpimaa, 2014. Ohjelmistolisenssit hallintaan – ja painajaiset pois. Viitattu 31.1.2017. http://www.talouselama.fi/tebatti/ohjelmistolisenssit-hallintaan-ja-painajaiset-pois-3451209

(14) Open Source Initiative. Viitattu 23.8.2017. https://opensource.org/faq

(15) Mikko Välimäki. Oikeudet tietokoneohjelmistoihin ja niiden lisensointi : Ohjelmisto-tuoteliiketoiminnan juridinen perusta. Helsinki: Turre Publishing; 2006. Viitattu 23.8.2017.

(16) Ben Balter, 2015. Copyright notices for open source projects. Viitattu 18.9.2017. https://ben.balter.com/2015/06/03/copyright-notices-for-websites-and-open-source-projects/